Universidad Zaragoza
1542

Trabajo Fin de Máster

Sensor autónomo inteligente para la evaluación del tráfico en una calzada

Intelligent autonomous sensor for traffic evaluation on a roadway

Autor/es

Belén Cebrián Martínez

Director/es

Julio David Buldain Pérez
Roberto Casas Nebra

Escuela de Ingeniería y Arquitectura

2019

Sensor autónomo inteligente para la evaluación del tráfico en una calzada

Resumen

El aumento de la congestión del tráfico en los últimos años ha supuesto la necesidad de mejorar las vías de transporte, la seguridad y la eficiencia del transporte terrestre, por lo que se ha desarrollado una red de transporte mediante la instalación de cámaras y sensores a lo largo de las vías que permite obtener la información necesaria sobre el tráfico. Para poder procesar toda la información recogida han surgido los Sistemas Inteligentes de Transporte (ITS), es decir, un conjunto de aplicaciones y sistemas tecnológicos que facilitan el control y la gestión del tráfico. Para el desarrollo de estos sistemas es necesario integrar dispositivos que permitan detectar vehículos y obtener información.

El sistema de monitorización del tráfico más utilizado en la actualidad es el bucle inductivo, pero proporciona poca información y tiene una vida útil limitada. Por este motivo se están desarrollando otros sistemas que permitan obtener mejores resultados.

El objetivo de este trabajo consiste en estudiar la posibilidad de utilizar un único sensor magnético para detectar el paso de los vehículos y estimar su tamaño, su velocidad y su dirección. Para ello, se pretende desarrollar un sistema que obtenga la información a partir de los cambios en el campo magnético terrestre detectados por el sensor magnético colocado bajo la calzada. Se pretende desarrollar también un sistema de reconocimiento de imágenes captadas por una cámara enfocada sobre la calzada para generar de forma automática las etiquetas de la base de datos que se utilizará para entrenar el sistema.

Inicialmente, se realiza un estudio sobre los sistemas más relevantes hoy en día para la monitorización del tráfico, así como las distintas líneas de desarrollo. La información obtenida en esta investigación se utilizará en la toma de decisiones para el diseño de los modelos clasificadores.

A continuación, se desarrolla un sistema automático para la creación de la base de datos. Para ello, se crea una infraestructura en la que los gestores se encargan de obtener los datos del sensor magnético y crear las etiquetas de los datos a partir de las imágenes captadas por la cámara. La base de datos creada se almacena en un servidor FTP.

Una vez creada la base de datos y a partir de la información obtenida de la investigación sobre los trabajos realizados en este campo, se diseña un clasificador binario para obtener la dirección de los vehículos y dos estimadores para obtener el tamaño y la velocidad.

Por último, se muestran los resultados obtenidos en este trabajo y se exponen las conclusiones y las posibles líneas de mejora que pueden utilizarse en trabajos futuros.

II

Índice general

Índice de figuras

Índice de tablas

Índice de ecuaciones

x

1 Introducción

1.1 Motivación y contexto

En los últimos años se ha producido un incremento de la congestión del tráfico debido al crecimiento de la población, reduciendo la eficiencia de la infraestructura de transporte e incrementando el tiempo de viaje, el consumo de combustible y la contaminación. Para mejorar las vías de transporte, aumentar la seguridad y eficiencia en el transporte terrestre y obtener la información necesaria sobre el tráfico se ha desarrollado una red de transporte mediante la instalación de cámaras y sensores a lo largo de las vías. La información recogida permite caracterizar el flujo de vehículos por las vías, detectar emergencias y notificarlas automáticamente, detectar las infracciones cometidas por los usuarios e incluso, estimar las trayectorias más probables para los vehículos mediante la re-identificación de vehículos a lo largo de las vías. Actualmente se recoge una gran cantidad de información a través de las redes de transporte que debe ser procesada para extraer la información útil, lo que ha propiciado la aparición de los Sistemas Inteligentes de Transporte (Intelligent Transportation System, ITS).

Se conoce como ITS al conjunto de aplicaciones y sistemas tecnológicos que facilitan el control y la gestión del tráfico con el objetivo de aumentar la seguridad. Estos sistemas se basan en redes de sensores distribuidos por las vías y proporcionan una gran información sobre el tráfico, ya que permiten identificar los tipos de vehículos que circulan por la vía, así como su tamaño o la ruta realizada. Para el desarrollo de ITS es necesario integrar dispositivos que permitan la detección de vehículos y la obtención de información. Sin embargo, en la actualidad ningún dispositivo es capaz de proporcionar toda la información necesaria, por lo que combinan distintos dispositivos con el fin de monitorizar la mayor información posible.

El sistema de monitorización del tráfico más utilizado hoy en día es el bucle inductivo, pero proporciona una información muy reducida y su vida útil es corta, por lo que se están desarrollando otros sistemas con una vida útil más larga que permitan obtener más información del tráfico. Por este motivo surge el presente Trabajo Final de Master (TFM), desarrollado en la Universidad de Zaragoza dentro del grupo de investigación HOWLab (Human Openware Research Lab [1] cuyo objetivo principal es la investigación y desarrollo de tecnologías centradas en las personas y sus entornos, Figura 1.1. Partiendo de un TFM desarrollado anteriormente [2], en este TFM se pretende obtener algunas de las principales características de los vehículos, tales como el tamaño o la velocidad, mediante un sensor magnético.

Figura 1.1: HOWLab, Universidad de Zaragoza

1.2 Objetivos del proyecto

En base al contexto planteado en el apartado anterior, el objetivo de este trabajo consiste en estudiar la posibilidad de detectar el paso de vehículos y obtener información de estos mediante un único sensor magnético. Para ello, se pretende desarrollar un sistema que, a partir de las señales obtenidas por el sensor magnético colocado bajo la calzada, sea capaz de detectar el paso de vehículos, así como estimar su tamaño, su velocidad y su dirección. El sistema debe ser autónomo e inteligente y con comunicación inalámbrica. Además, para realizar el entrenamiento del sistema será necesario un sistema de reconocimiento de imágenes captadas por una cámara enfocada sobre la calzada para generar de forma automática las etiquetas para las señales captadas de los vehículos.

Para llegar a alcanzar este objetivo, se proponen los siguientes objetivos parciales del trabajo:

1. Estudio sobre los diferentes métodos de clasificación de vehículos.
2. Desarrollo de un sistema de reconocimiento de imágenes y generación de etiquetas.
3. Desarrollo de un gestor encargado de la comunicación con el sensor y de procesar los datos recibidos del sensor.
4. Creación de la base de datos formada por las etiquetas y las huellas magnéticas.
5. Entrenamiento del sistema para identificar las características de los vehículos en las huellas magnéticas.
6. Validación de la calidad del sistema entrenado.

1.3 Metodología

Para realizar este proyecto se parte de la hipótesis de que la información que proporciona la huella magnética de un vehículo obtenida con un único sensor es suficiente para extraer su tamaño, velocidad y dirección. Para lograr el objetivo principal del trabajo, se decide desglosar el problema de manera que se vaya cumpliendo cada uno de los objetivos parciales. De esta forma, se descompone el trabajo en diferentes fases.

La primera fase del trabajo será realizar un estudio sobre las diferentes tecnologías y técnicas utilizadas para la clasificación de vehículos.

La siguiente fase consistirá en colocar una cámara enfocada a la zona de la calzada en la que está colocado el sensor y programada para que grabe cuando detecte movimiento en la zona y suba los videos a un servidor FTP. A continuación, se desarrollará un sistema de reconocimiento de imágenes automático, siendo capaz de obtener los videos del servidor y procesarlos para generar las etiquetas de los vehículos.

Seguidamente, se desarrollará un gestor que permita la comunicación con el sensor. Se encargará también del procesamiento de los datos recibidos por el sensor y de subirlos al servidor FTP.

Una vez se tenga el sistema listo, se procederá a la recogida de datos de los vehículos que pasen por la zona del sensor y se creará la base de datos a partir de las huellas magnéticas y las etiquetas generadas.

Por último, se realizará un análisis para encontrar el sistema óptimo para estimar las características de los vehículos y se entrenará el sistema elegido.

1.4 Estructura de la memoria

Este documento se divide en dos partes: la primera parte contiene la memoria del proyecto, en la que se resume el trabajo realizado; y la segunda contiene los anexos que explican con mayor detalle ciertos aspectos de la memoria.

La memoria contiene los siguientes capítulos:

- Capítulo 2 – Estado del arte: contiene un análisis sobre los distintos sistemas que se utilizan actualmente para la detección y clasificación de vehículos.
- Capítulo 3 – Base de datos: contiene la creación de la infraestructura para la creación automática de la base de datos.
- Capítulo 4 – Modelo de clasificación: se realiza un tratamiento de la base de datos y se desarrollan los clasificadores para la detección de los vehículos.
- Capítulo 5 – Conclusiones y trabajo futuro: se recogen las conclusiones del proyecto y el trabajo futuro a realizar.

Los anexos son los siguientes:

- Anexo A: Resultados de las pruebas realizadas en el tratamiento de los datos.
- Anexo B: Resultados obtenidos durante el desarrollo de los clasificadores.
- Anexo C: Cronograma de la planificación temporal
- Anexo D: Código del gestor de las imágenes de la cámara.
- Anexo E: Código del gestor de los datos del sensor.
- Anexo F: Código para actualizar la base de datos.
- Anexo G: Código para el tratamiento de los datos.
- Anexo H: Código del clasificador.
- Anexo I: Código del estimador del tamaño.
- Anexo J: Código del estimador de la velocidad.

2 Estado del arte

En este capítulo se realiza un estudio sobre los sistemas que se encuentran en uso actualmente para la detección de vehículos y la obtención de información de estos.

2.1 Sistemas basados en bucles inductivos

Uno de los sistemas más utilizados actualmente consiste en detectar las variaciones de inductancia de una espira cuando pasa sobre ella un objeto metálico de gran tamaño. Para ello, se introduce una espira de alambre en el pavimento de la carretera que, al pasar un vehículo sobre ella, detecta los cambios de inductancia generando una señal que varía en función de la clase de vehículo (Figura 2.1).

Figura 2.1: Sistema basado en bucle inductivo [3]

Para tratar los datos obtenidos por el bucle inductivo se han propuesto diferentes métodos con el objetivo de obtener la mayor información sobre el tráfico con la mejor precisión posible. Algunos trabajos utilizan las redes neuronales para clasificar los vehículos según el tamaño. S. Meta y M. G. Cinsdikici [3] proponen una técnica basada en una red neuronal multicapa (Backpropagation Neural Network, BPNN) para clasificar los vehículos en 5 clases según su tamaño: coche, furgoneta, camión, autobús y motocicleta. Para obtener las señales de entrada de la red neuronal proponen procesar los datos de la siguiente manera: primero realizan la transformada discreta de Fourier (Discrete Fourier Transform, DFT) de los datos recibidos y, a continuación, reducen su dimensionalidad con un análisis de componentes principales (Principal Component Analysis, PCA). Con este método consiguen una eficacia de 94.21%.

Y. Ki y D. Baik [4] proponen también la utilización de una red neuronal (Artificial Neural Network, ANN), pero utilizan como características principales el ratio de variación de la frecuencia y la forma de onda de la frecuencia. En este caso consiguen una eficacia de 91.5%.

Por otro lado, Y. Lao *et al* [5] utilizan el modelo mixto gaussiano (Gaussian Mixture Model, GMM) para estimar la velocidad de los vehículos y clasificarlos según el tamaño. Para evaluar la eficiencia del método utilizan el error absoluto medio (average absolute error, AAE) y la proporción de clasificaciones correcta, obteniendo una precisión del 97.6%.

Este sistema obtiene información limitada sobre los vehículos, por lo que se ha desarrollado una versión en la que se colocan dos espiras separadas una distancia conocida de manera que, cuando un vehículo pasa por la primera espira, se cuenta el tiempo que tarda en pasar sobre la segunda. De esta forma se puede obtener la velocidad y el tamaño del vehículo con mayor precisión, sin embargo, este sistema es

más costoso y su vida útil es inferior que la del sistema de 1 bucle inductivo.

J. Gajda y M. Stencel [6] proponen un método basado en un detector de bucle inductivo doble para clasificar vehículos de mercancías TT (2+3). El método consiste en que cada clase tiene implementado un algoritmo diferente, ya que cada algoritmo utiliza su propio conjunto de parámetros y rangos de variabilidad y extrae las características del vehículo de la señal recibida que son más relevantes para esa clase. Para clasificar los vehículos se basa en el número de ejes y la distancia entre ellos. El método de clasificación consiste en un árbol de decisión en el que comprueba si el vehículo cumple con las características de una clase. Si las cumple se le asigna esa clase y si, por el contrario, no las cumple se pasa a comprobar las características de la siguiente clase. Con este método consigue una eficacia de 90%.

2.2 Sistema de reconocimiento visual de vehículos

Los sistemas de reconocimiento visual están siendo muy utilizados dentro de los ITS debido a que proporciona muy buenos resultados y su instalación es sencilla. En la actualidad se dispone de una gran red de cámaras distribuidas por la mayoría de las vías que permite tener un control del estado del tráfico, contabilizar los vehículos que pasan por la vía o identificar y notificar un problema en la vía, entre otras cosas. Para poder gestionar toda la información proporcionada por las cámaras es necesario un sistema automático de reconocimiento visual.

Como se observa en la Figura 2.2, a partir de los datos proporcionados por las cámaras, se busca obtener la mayor información posible sobre el tráfico, como la cantidad de vehículos de cada clase que circulan por la vía, sus velocidades o la re-identificación de vehículos, obtenida mediante un seguimiento del vehículo a través de la red de cámaras, permitiendo predecir el flujo de tráfico en las vías.

Figura 2.2: Reconocimiento visual de vehículos [7]

A lo largo de estos años se han desarrollado numerosas investigaciones para obtener la máxima precisión posible en la clasificación de los vehículos. A. Ghosh et al. [8] propone un sistema de procesamiento de imagen digital tanto para videos grabados como para videos en tiempo real basado en OpenCV (Open Source Computer Vision Library). Para ello, convierte el video en una secuencia de imágenes, extrae la imagen de referencia y realiza la detección de objetos moviéndose. Si el área del objeto en movimiento es mayor que el umbral fijado, el objeto se considera un vehículo. Además, estima la velocidad

del vehículo a partir del tiempo que tarda en cruzar las líneas de referencia del sistema. Este sistema consigue una eficacia de 80%.

Un gran número de los proyectos que utilizan reconocimiento visual para la clasificación de vehículos proponen utilizar redes neuronales debido a que estos sistemas necesitan una potencia de cálculo elevada y existen muchas variables que dificultan la correcta clasificación de los vehículos, por ejemplo, las condiciones atmosféricas. S. Zhang et al. [9] extrae las características más importantes a través de redes neuronales convolucionales (Convolutional Neural Network, CNN), histogramas de gradientes orientados (Histograms of Oriented gradients, HOG) y análisis de componentes principales (PCA) para, posteriormente, clasificar los vehículos utilizando una máquina de vectores soportes (Support Vector Machine, SVM). Con este método obtiene una eficacia de 98%. Por otro lado, J. Sochor et al. [7] propone utilizar cajas delimitadoras 3D obtenidas de los vehículos para identificar las características del vehículo y utilizar CNNs pre-entrenadas. De esta forma consigue una precisión de 94.6%. Por último, X. Le et al. [10] utiliza imágenes obtenidas de un dron y las características aprendidas de CNNs pre-entrenadas y las combina con las características obtenidas a mano para mejorar la precisión de la clasificación para vehículos pequeños. Este sistema clasifica en 4 clases con una eficiencia de 80%.

Otros proyectos permiten identificar los vehículos y realizar un seguimiento a lo largo de las vías a partir del reconocimiento de las matrículas, como proponen E. C. Neto et al. [11] Para ello realizan primero una fase de procesado de los datos en los que aplican un detector de bordes Canny y la transformada de Hough y, a continuación, realizan la identificación de los números y las letras a partir de un Perceptrón Multicapa (MultiLayer Perceptron, MLP) junto con una máquina de aprendizaje extremo (Extreme Learning Machine, ELM) y mínimos cuadrados (Least Squares, LS). Este sistema tiene una eficacia de 97% en la extracción y reconocimiento de números y una eficacia de 96.7% en el reconocimiento de letras, por lo que el sistema final tiene una eficacia de 94%.

2.3 Sistemas basados en señales acústicas

Para realizar la clasificación de vehículos se pueden emplear las diferentes señales que emiten los vehículos, como el campo magnético, sonidos, ruido o calor, entre otros. Cuando un vehículo está en movimiento emite sonidos característicos generados por el movimiento de algunas partes, fricciones, viento, emisiones, etc. Asumiendo que estos sonidos son iguales para vehículos similares en condiciones similares, es posible utilizarlos para realizar la clasificación.

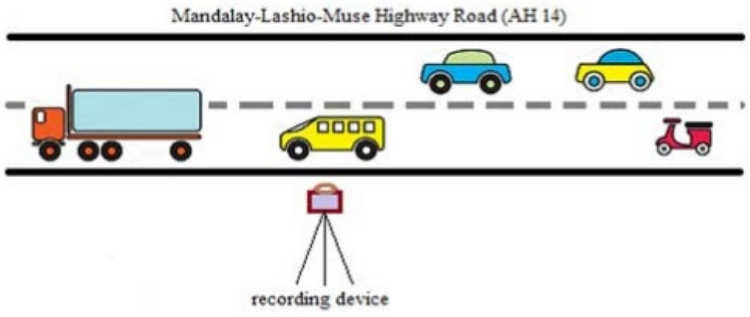


Figura 2.3: Esquema del sistema [12]

Procesar las señales acústicas para realizar la clasificación supone menor coste, carga computacional y tiempo de consumo que las señales de video. Por este motivo, se han desarrollado diversas investigaciones basadas en estas señales. En su proyecto, L. N. Thu, A. Win y H. N. Oo [12] colocan una videocámara al lado de la carretera que graba los sonidos que emiten los vehículos al pasar, Figura 2.3. Además, utilizan el video grabado por la cámara para generar las etiquetas de los datos obtenidos para crear la base de datos. Proponen realizar la extracción de características utilizando Coeficientes Cepstrales en las Frecuencias de Mel (Mel-Frequency Cepstral Coefficient, MFCC) y utilizar la red ConvNet para realizar la clasificación. Con este sistema consiguen clasificar los vehículos en 4 clases con una eficacia de 85%.

A. D. Mayvan, S. A. Beheshti y M. H. Masoom [13] utilizan como características principales de los datos la energía a corto plazo (Short-time Energy), la velocidad de cruce por cero (Short-time average Zero Cross Rate) y la frecuencia de tono (Pitch). Estas características se usan para la fase de entrenamiento y clasificación, en la cual proponen utilizar el Análisis Discriminante Cuadrático (Quadratic Discriminant Analysis, QDA), consiguiendo una eficacia de 69%. Por último, M. P. Paulraj et al. [14] diseñan un sistema basado en el principio de Doppler, que describe que el pitch y la frecuencia incrementan cuando el sonido que emite el objeto se acerca al observador y disminuye cuando se aleja. Para la extracción de características utilizan un modelo autorregresivo y la clasificación se realiza a través de una red neuronal probabilística (Probabilistic Neural Network, PNN), consiguiendo clasificar los vehículos según el tipo con una eficacia de 94% y estimar la distancia a la que se encuentran con una eficacia de 92%.

2.4 Sistemas basados en fibra óptica

En los últimos años se han desarrollado sistemas basados en fibra óptica para el control del tráfico terrestre ya que proporcionan muy buenos resultados. J. Nedoma et al. [15] proponen un sistema para detectar los vehículos que pasan basado en un interferómetro Mach-Zehnder. Para ello, el sistema se coloca en el arcén, figura 2.4, detecta las vibraciones generadas por los vehículos al pasar junto al sensor. Además, se caracteriza por su inmunidad a la interferencia magnética (EMI) y por su simple implementación, ya que requiere instalación en la carretera. La eficacia del sensor depende de la distancia a la que se encuentra de la carretera, de manera que al alejarlo disminuye la eficacia. De esta forma, consiguen una eficacia de 98.14% colocando el sistema a 1 metro y de 56.14% colocándolo a 3 metros.

Figura 2.4: Prototipo final del sensor [15]

En otro proyecto, J. Nedoma et al. [16] utilizan dos interferómetros colocados a una distancia conocida y, a partir del espacio temporal entre las señales recibidas por ambos sensores y la distancia entre ellos, estiman la velocidad del vehículo con un error absoluto máximo de 1.53 kph.

Por otro lado, M. Bin y Z. Xinguo [17] proponen un sistema basado en fibra óptica para calcular el peso de los vehículos en movimiento. Cuando un vehículo pasa sobre el sensor ejerce una presión que provoca una reducción de la intensidad de la luz recibida, que es independiente de los cambios producidos por la fuente de luz, permitiendo estimar el peso del vehículo. Las ventajas de este sistema son su alta precisión, una vida útil larga y que no le afectan las interferencias electromagnéticas.

2.5 Sistemas basados en sensores magnéticos

Por último, otra de las señales que se pueden medir para realizar la clasificación de vehículos es la variación en el campo magnético terrestre. El campo magnético terrestre es prácticamente constante en la superficie de la tierra, pero en presencia de un objeto metálico de gran tamaño se producen perturbaciones en el campo. Estas perturbaciones son detectadas por los sensores magnéticos que, además de detectar los vehículos, obtienen una huella magnética de cada vehículo, la cual depende de la cantidad de metal que tiene el vehículo, el tamaño, la velocidad y la posición. Los sistemas basados en sensores magnéticos proporcionan más información y tienen una vida útil mayor que otros sistemas basados en distintas tecnologías, como los bucles inductivos.

Estos sistemas se integran en las vías de manera que miden los cambios en el campo magnético cuando pasa un vehículo sobre el sensor y, a partir de esas señales, se extraen las características del vehículo.

Figura 2.5: Sistema basado en sensor magnético [18]

S. Vançin y E. Erdem [18] proponen un sistema basado en un sensor magnetorresistivo anisotrópico (anisotropic magnetoresistive sensor, AMRs) para detectar el estado del tráfico, clasificar vehículos y detectar la dirección de los vehículos. Para detectar el estado del tráfico, colocan 3 sensores a lo largo de 100 metros en la carretera, figura 2.5, los cuales mandan las señales magnéticas a un nodo coordinador que transmite las señales a un ordenador en el que se procesan los datos. Con este sistema evalúan la densidad del tráfico en cuatro categorías: si todos los sensores detectan vehículos al mismo tiempo, lo identifica como mucho tráfico; si no se detecta ningún vehículo, lo identifica como no tráfico; si dos sensores detectan vehículos, lo identifica como tráfico; y si solo un sensor detecta vehículo, lo identifica como poco tráfico.

Por otro lado, a partir de la longitud de la firma magnética realizan una clasificación de los vehículos en cuatro categorías: coche, minibús, bus y camión, obteniendo una precisión de 95%. Por último, detectan si el vehículo se desplaza de izquierda a derecha, o viceversa, utilizando dos sensores colocados a cinco metros de distancia, consiguiendo una precisión de 94%.

G. Burresi y R. Giorgi [19] utilizan dos sensores magnéticos separados una distancia de 1.2 metros y colocados en la misma orientación. Para realizar la clasificación utilizan un perceptrón multicapa (Multi-Layer Perceptron, MLP) con dos capas ocultas y función de activación sigmoidea. Este sistema consigue clasificar los vehículos en cuatro tipos con una precisión de 96.7%.

Otros autores se centran en desarrollar un sistema basado en un único sensor magnético consiguiendo una simplificación del sistema y una reducción de los costes. Siguiendo esta línea, X. Chen [20] utiliza un sensor AMR de tres ejes para medir los cambios en el campo magnético terrestre cuando pasa un vehículo por la zona experimental. Para ello, coloca el sensor en el arcén en una posición fija en el suelo, por lo que solo puede detectar un vehículo al mismo tiempo y la separación entre vehículos tiene que ser de al menos un segundo. Además, utiliza los Coeficientes Cepstrales en las Frecuencias de Mel (Mel Frequency Cepstral Coefficients, MFCC) para extraer las características de las señales magnéticas y la Deformación dinámica del tiempo (Dynamic Time Warping, DTW) y la Cuantificación vectorial (Vector Quantization, VQ) para la clasificación en cuatro tipos de vehículos: sedan, furgoneta, camión y bus. De esta forma, consigue una eficacia de 90%.

Y. He, Y. Du y L. Sun [21] proponen un sistema basado en un único nodo con dos sensores magnéticos colocado en el lateral de la carretera. Además, utilizan un modelo Filter-Filter-Wrapper para obtener el conjunto de características óptimo. Con este conjunto de características proponen un algoritmo de clasificación basado en máquinas de vectores de soporte de agrupamiento (Clustering Support Vector, C-SVM). Para ello, desarrollan un árbol de decisión binario que clasifica dos tipos de vehículos en cada paso mediante el clasificador C-SVM. En el primer paso, clasifican en coche pequeño-mediano y coche grande y, en el segundo paso, clasifican en buses y camiones. Con este método consiguen una eficacia de 92.80%.

Por último, R. Bhattarya [22] propone un sistema de clasificación de vehículos mediante redes neuronales (Machine-Learning Vehicle Classification, MLVC) con un sensor AMR. Realiza una comparativa entre el clasificador CART (Classification And Regression Tree) y el perceptrón multicapa MLP, obteniendo en ambos casos una eficacia mayor de 90%.

2.6 Conclusiones

En este capítulo se han presentado los principales sistemas para la detección de vehículos utilizados actualmente y las distintas investigaciones realizadas en este campo. A partir de la comparativa realizada en la tabla 2.1 se puede concluir que no es posible decidir a priori qué sistema es más eficaz para la clasificación de los vehículos, ya que cada autor utiliza su propia base de datos, por lo que las dimensiones de las bases de datos utilizadas no son iguales, y clasifican los vehículos en distintos grupos. Sin embargo, se decide utilizar clasificadores basados en redes neuronales debido a los buenos resultados obtenidos en las distintas investigaciones. Por otro lado, se decide utilizar para el desarrollo del TFM las huellas magnéticas obtenidas de un sensor magnético colocado bajo la calzada.

Autores	Tecnología de sensado	Técnica de reconocimiento	Tamaño de la base de datos	Eficacia
S. Meta y M. G. Cinsdikici [3]	Bucles inductivos	BPNN	2330	94.21%
Y. Ki y D. Baik [4]	Bucles inductivos	ANN	1055	91.5%
Y. Lao *et al* [5]	Bucles inductivos	GMM	21591	97.6%
J. Gajda y M. Stencel [6]	Bucles inductivos	Árbol de decisión	244	90%
J. Sochor et al. [7]	Reconocimiento visual	CNN pre-entrenadas	21250	94.6%
A. Ghosh et al. [8]	Reconocimiento visual	OpenCV	500	80%
S. Zhang et al. [9]	Reconocimiento visual	SVM	4716	98%
X. Le et al. [10]	Reconocimiento visual	CNN pre-entrenadas	1792	80%
E. C. Neto et al. [11]	Reconocimiento visual	MLP y ELM	45270	94%
L. N. Thu, A. Win y H. N. Oo [12]	Señales acústicas	ConvNet	402	85%
A. D. Mayvan, S. A. Beheshti y M. H. Masoom [13]	Señales acústicas	QDA	240	69%
M. P. Paulraj et al. [14]	Señales acústicas	PNN	540	92%
J. Nedoma et al. [15]	Fibra óptica	1 interferómetro Mach-Zehnder	504	98.14%
J. Nedoma et al. [16]	Fibra óptica	Distancia entre 2 interferómetros Mach-Zehnder	100	Error máximo de 1.53 kph
M. Bin y Z. Xinguo [17]	Fibra óptica	Detector de cambios de intensidad de la luz	-	90%
S. Vançin y E. Erdem [18]	Sensor magnético	Longitud de la firma magnética y distancia entre los sensores	100	Clasificación: 95% Dirección:94%
G. Burresi y R. Giorgi [19]	Sensor magnético	MLP	332	96.7%
X. Chen [20]	Sensor magnético	DTW y VQ	148 y 291	90%
Y. He, Y. Du y L. Sun [21]	Sensor magnético	Árbol de decisión binario + C-SVM	460	92.80%
R. Bhattarya [22]	Sensor magnético	CART y MLP	5760	>90%

Tabla 2.1: Resumen de todos los métodos

3 Sistema de creación de base de datos

En este capítulo se explica el proceso de creación de la base de datos. Para ello, se describe la infraestructura desplegada y el sistema propuesto para la captura de los datos y el procesado de estos. Se describe también el procesado utilizado para crear las etiquetas de los datos.

3.1 Descripción del sistema

Para crear la base de datos se ha montado una infraestructura que permite recoger datos de forma automática. Esta infraestructura se puede ver en el esquema de la figura 3.1. El sistema está formado por un sensor magnético que envía los datos de forma inalámbrica utilizando LoRaNet [23] a un gestor encargado de recibirlos, procesarlos y enviarlos a un cliente InfluxDB [25] donde se almacenan y se representan automáticamente, por otro lado, una cámara enfocada en la zona del sensor envía los videos de los vehículos que pasan a un servidor FTP a través de una red WiFi. Un gestor se encarga de obtener los videos del servidor y procesarlos para extraer las características de los vehículos y crear las etiquetas, las cuales envía al servidor FTP. Por último, otro gestor se encarga de obtener los datos del cliente InfluxDB e identificar los datos que corresponden a huellas magnéticas de vehículos. Cuando detecta una huella magnética comprueba si existe una etiqueta en el servidor FTP que coincida con la hora asociada a esa huella, por lo que el sistema debe estar perfectamente sincronizado. Si encuentra la etiqueta de esa huella, actualiza la base de datos almacenada en el servidor FTP con la huella y la etiqueta.

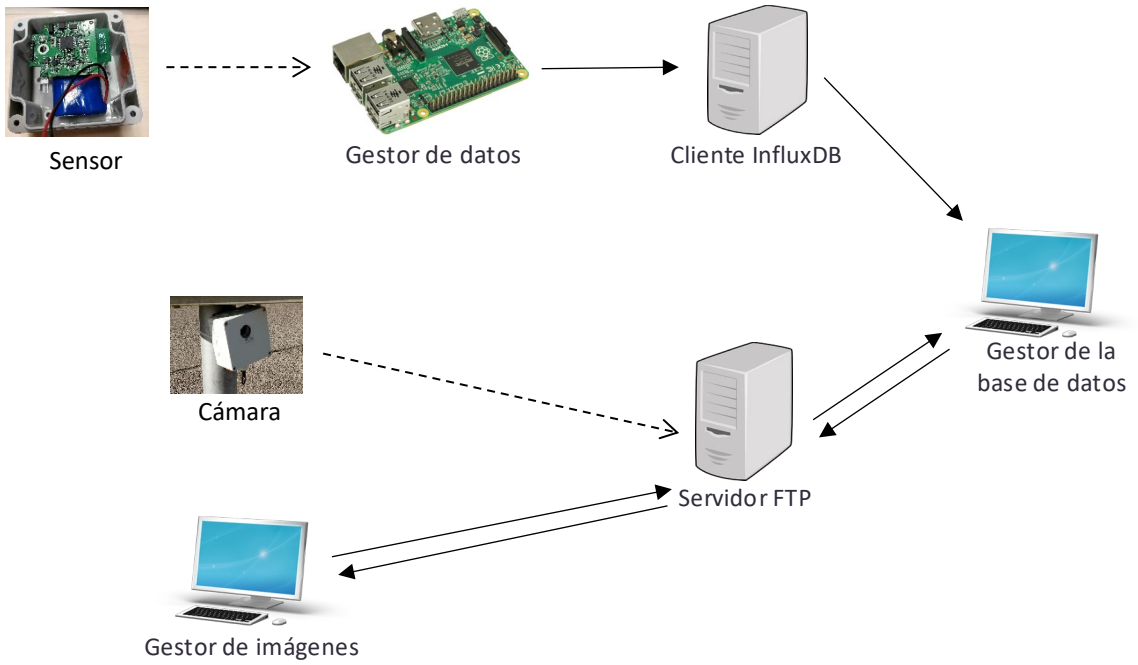


Figura 3.1: Infraestructura para crear la base de datos

Para obtener los videos se utiliza una cámara IP que se conecta a la red WiFi y que ha sido configurada para que grabe únicamente cuando hay una alarma de movimiento en la zona del sensor. De esta forma, cuando se activa la alarma de movimiento la cámara graba un video de duración inferior a un minuto y lo manda a un servidor FTP.

Por otro lado, el sensor utilizado es un prototipo formado por un acelerómetro y magnetómetro, un módulo de comunicación inalámbrica LoRaNet, una memoria flash de 64Mb y un microcontrolador de 16 bits. El sensor muestrea los datos recogidos por el magnetómetro a 100 Hz y los envía por LoRaNet en paquetes de 15 mensajes de 96 bytes que contienen 48 datos, 16 datos por eje, cada uno. Estos mensajes los recibe una Raspberry Pi con un módulo de comunicación LoRaNet encargada de extraer los datos de los mensajes y mandarlos al servidor FTP.

Figura 3.2: Ubicación inicial del sistema

La ubicación inicial del sistema se muestra en la figura 3.2. Consistía en colocar el sensor bajo la calzada en mitad del carril de salida detrás del edificio Ada Byron de la Universidad de Zaragoza y colocar el gestor de los datos y la cámara en el laboratorio 4.01 del edificio, de manera que la cámara enfocase la zona del sensor. Durante las pruebas iniciales surgen problemas de comunicación con el sensor, provocando que no sea posible acceder a su configuración. Por este motivo, se decide cambiar la ubicación del sensor y, por tanto, la ubicación de la cámara. Finalmente, se decide colocar el sensor en la rampa de entrada/salida del aparcamiento del Instituto de Investigación en Ingeniería de Aragón (I3A) ya que se dispone de una caja estanca instalada previamente en la calzada. La cámara se coloca en un poste en el lateral de la rampa, a unos 8 metros de distancia del sensor, y el gestor de datos se deja en el laboratorio 4.01. En la figura 3.3 se puede observar la ubicación final del sistema.

Figura 3.3: Ubicación actual del sistema

3.2 Procesamiento de los datos

Para obtener la base de datos se realizan tres fases. Por un lado, se generan las etiquetas a partir de los videos de los vehículos que pasan por la zona del sensor. Por otro lado, se obtienen los datos que manda el sensor y, por último, se juntan los datos y las etiquetas dando lugar a la base de datos.

3.2.1 Generación de las etiquetas

Como se ha mencionado anteriormente, cuando un vehículo pasa por la zona del sensor la cámara genera un vídeo de duración inferior a un minuto y lo manda al servidor FTP. El gestor de imágenes se encarga de procesar los datos y crear las etiquetas. En la figura 3.3 se muestra el diagrama de flujo que ejecuta el gestor para procesar los videos y generar las etiquetas. En el Anexo D se encuentra el código del gestor.

Cuando hay un nuevo video en el servidor FTP el gestor lo descarga y, si no está dañado, comienza a procesar el video utilizando la librería de OpenCV [25]. En esta fase obtiene la imagen de fondo a partir del primer frame del video, figura 3.5.a, y se la resta a cada frame, figura 3.5.b. De la imagen resultante, figura 3.5.c, se obtienen los blobs. De cada blob, si es mayor que el área mínima correspondiente a un vehículo, se extrae el tamaño y se comprueba si es la primera vez que aparece en el video. Si es la primera vez, se registra como un nuevo vehículo y se guarda la fecha y hora en la que llega a la zona del sensor, delimitada por las líneas verdes en la figura 3.5.d. Si corresponde a un vehículo identificado anteriormente se obtiene la dirección y la velocidad, obtenida a partir del flujo óptico, y se actualizan sus características. Cuando se termina de procesar el video se crean las etiquetas de los vehículos detectados, figura 3.6, y se mandan al servidor FTP junto con las partes del video en las que pasan los vehículos.

En la ubicación inicial de la cámara surgía un problema con la sombra de los vehículos puesto que el gestor no es capaz de distinguir su sombra, obteniendo una medida errónea del tamaño del vehículo. En la nueva ubicación del sistema se ha resuelto este problema excluyendo el suelo de la imagen que se procesa, como se puede observar en las imágenes de la figura 3.5.

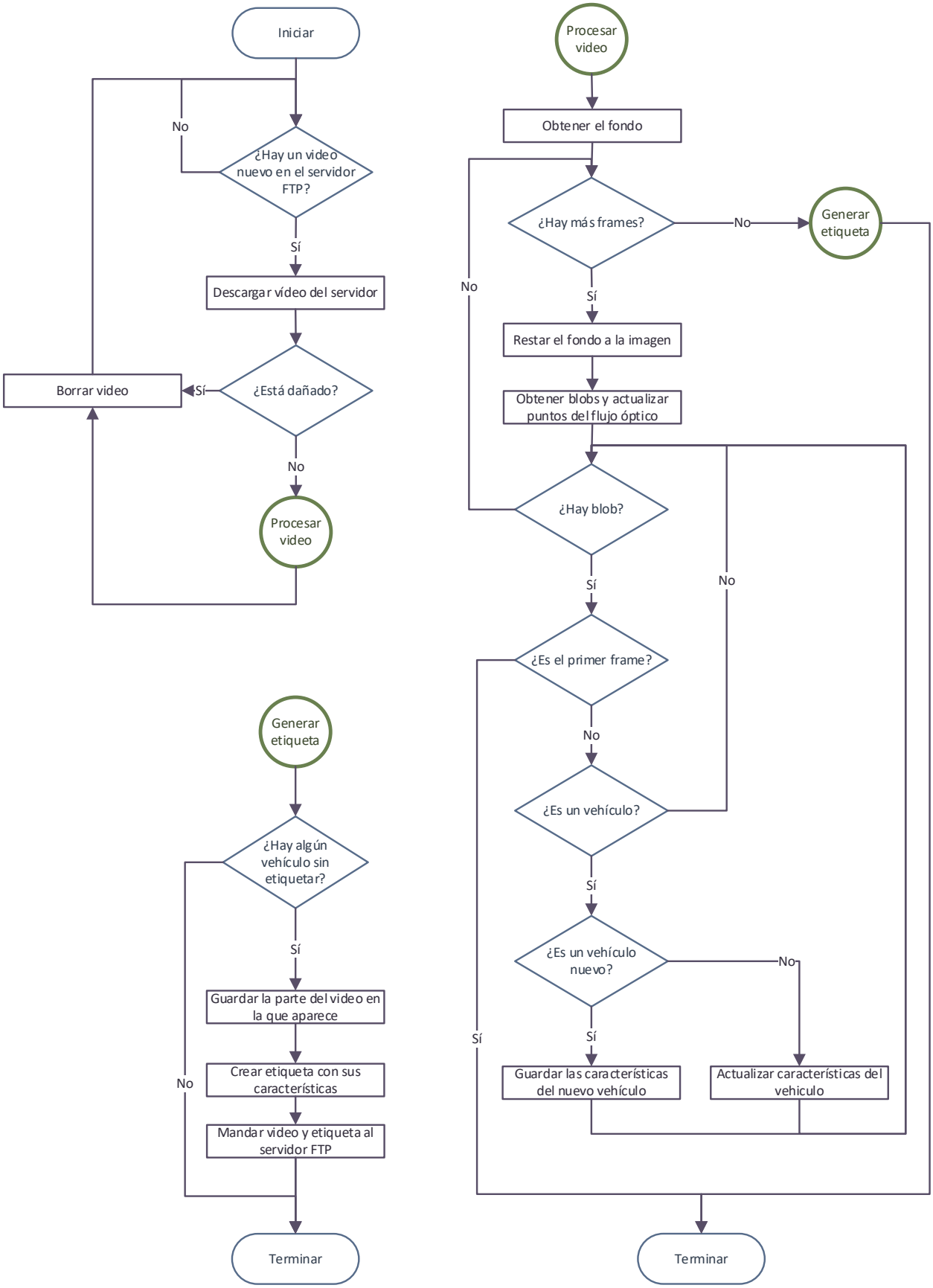
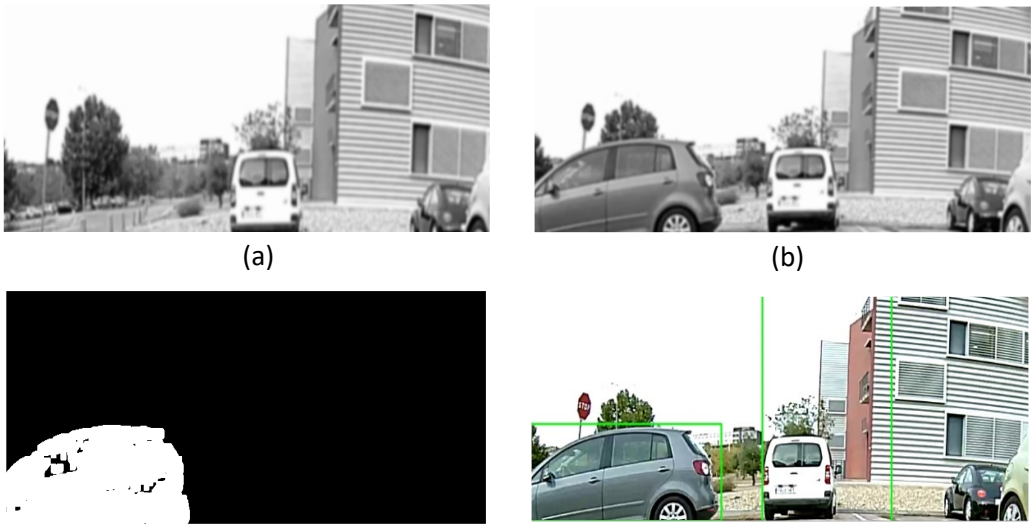


Figura 3.4: Diagrama de flujo del gestor de imágenes



(a) (b)

(c) (d)

Figura 3.5: (a) imagen de fondo (b) frame en escala de grises (c) diferencia entre fondo y frame (d) Detección del vehículo


```
{
    "vehiculo": [{
            "Timestamp inicio": 1572445458,
            "Timestamp fin": 1572449059,
            "ID video": "1572445458.mp4",
            "Tamagno (metros)": 3.85,
            "Velocidad (km/h)": 13.24,
            "Direccion": "sale"
        }
    ]
}
```


Figura 3.6: Etiqueta del vehículo

3.2.2 Procesamiento de los datos del sensor magnético

Como se ha explicado anteriormente, el sensor magnético se coloca en la rampa de entrada/salida del aparcamiento y el gestor de los datos en el laboratorio 4.01 del edificio Ada Byron. En esta nueva ubicación el sistema no puede estar fijo porque la cámara no está preparada para los fenómenos medioambientales, como la lluvia o el viento, por lo que el sistema se coloca en cada recogida de datos. El sensor se coloca siempre en la misma posición, de manera que la dirección de los ejes del magnetómetro es siempre la misma, siendo ésta la mostrada en la figura 3.7.

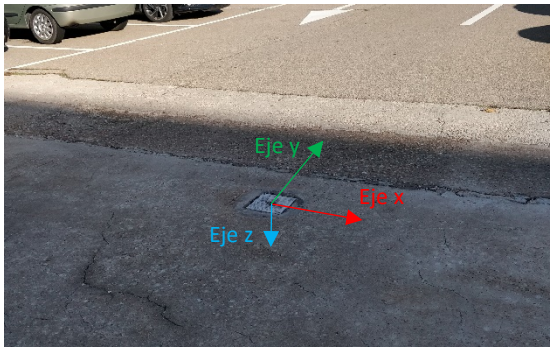


Figura 3.7: Orientación del sensor

Inicialmente, el sensor comparaba la señal del magnetómetro con un umbral, mandando datos únicamente cuando se superaba dicho umbral, mientras que el gestor se encargaba de recibir los paquetes, extraer los datos y mandarlos al servidor FTP y a un cliente InfluxDB que almacena los datos y los representa automáticamente. Tras realizar una batería de pruebas se observó que con este método se estaba perdiendo parte de la forma de onda de la huella magnética del vehículo, provocando la pérdida de información, por lo que se decidió modificar el sensor para que envíe datos constantemente y el gestor para que reciba los mensajes, extraiga los datos y los mande al cliente InfluxDB, donde se almacenan, para decidir posteriormente qué datos corresponden a las huellas magnéticas de vehículos y crear la base de datos con las etiquetas.

Por tanto, el gestor desarrolla dos funciones, por un lado, permite la comunicación con el sensor de forma remota a través de un socket TCP/IP, de manera que cualquier cliente desde cualquier lugar puede acceder a la configuración LoRaNet del sensor y comunicarse con él a través del gestor. Para ello, el gestor crea un socket TCP/IP con una IP pública y, cuando se establece una conexión, actúa de puente en la comunicación entre el cliente y el sensor. Además, se han desarrollado un conjunto de funciones que simplifican los mensajes que debe mandar el cliente. Estas funciones son:

- getParams(): obtiene una lista de los registros de la configuración de LoRaNet del sensor.
- setParams(*args, **kwargs): modifica el valor de los registros que se pasan como parámetros.
- establishNetwork(): crea una red a partir de los parámetros de los registros, recluta los nodos disponibles y envía al cliente una lista con los nodos de la red y sus direcciones.
- destroyNetwork(): destruye la red existente.
- updateNetwork(): comprueba el estado de la red, actualiza la lista de nodos y recluta nuevos nodos disponibles.
- sendMessage(addr, XX, message): envía el mensaje unicast de tamaño XX al nodo con dirección addr.
- sendBroadcast(message): envía el mensaje broadcast a todos los nodos de la red.
- sendBeacon(message): envía el beacon a todos los nodos que puedan escucharle.

Por otro lado, el gestor recibe los beacon que envía el sensor y extrae los datos formados por las muestras de cada eje y la hora en la que se recibe. Como se muestra en la figura 3.8, cuando el gestor recibe un beacon con el formato esperado comprueba el identificador del mensaje. Si el identificador no es el siguiente al último recibido significa que los mensajes con los identificadores comprendidos entre el último recibido y el del nuevo beacon se han perdido, por lo que completa estos datos con ceros. Si el identificador es 0 se guarda la hora en la que ha recibido el paquete para calcular el timestamp de cada dato utilizando la frecuencia de muestreo. A continuación, decodifica los datos del mensaje y envía al cliente InfluxDB un paquete formado por las muestras de los ejes y sus timestamp. El código completo del gestor se encuentra en el Anexo E.

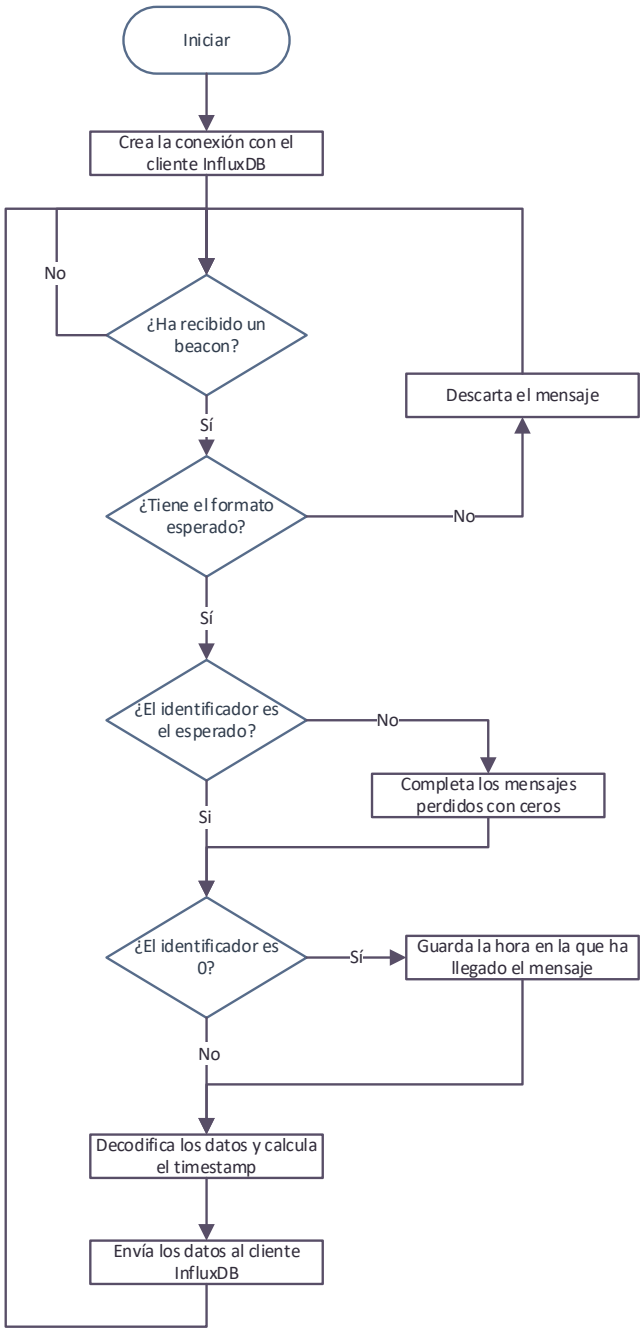


Figura 3.8: Diagrama de flujo de la recogida de datos

3.2.3 Actualización de la base de datos

Al finalizar la recogida de datos se procede a la actualización de la base de datos a partir de los videos cortados de los vehículos y las etiquetas disponibles en el servidor FTP y de los datos del sensor almacenados en el cliente InfluxDB. Para ello, se descarga del cliente InfluxDB un fichero con todos los datos almacenados durante la recogida y el gestor de la base de datos se encarga de decidir qué datos corresponden a las huellas magnéticas de vehículos y de mandarlos al servidor FTP.

Una vez descargado el fichero con los datos, el gestor carga el fichero y resta la media de cada eje, con el objetivo de eliminar el offset que puedan tener las señales. A continuación, recorre las muestras comparándolas con un umbral de ±50mG. Se considera que es una huella de vehículo cuando las muestras de los tres ejes superan el umbral hasta que dejan de superarlo. Para evitar perder parte de la forma de onda de las señales, como pasaba anteriormente, se toma como inicio de la huella 20 muestras antes de superar el umbral y como final 20 muestras sin superar el umbral. Además, para evitar confundir una huella con un cambio repentino en el campo magnético, comprueba que la longitud del patrón recogido sea superior a 60 muestras, de manera que al menos 20 muestras han superado el umbral. A cada huella se le asigna un patrón con las muestras de los tres ejes y los timestamp de cada muestra.

Una vez extraídas las huellas magnéticas, compara el timestamp de inicio de cada huella con los timestamp de las etiquetas extraídas de los videos. Dado que el sistema está sincronizado, si hay una etiqueta con el mismo timestamp que la huella con un margen de ±1 segundos genera un fichero con la huella magnética y lo envía a la base de datos ubicada en el servidor FTP y mueve la etiqueta y el video correspondientes a la base de datos. El código utilizado se encuentra en el Anexo F.

3.3 Conclusiones y limitaciones

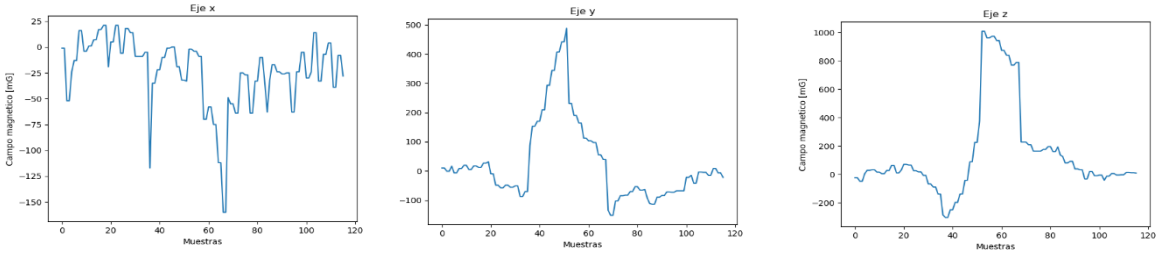
La infraestructura creada permite obtener una base de datos de forma automática, de manera que una vez se colocan la cámara y el sensor, el sistema actualiza la base de datos con los nuevos vehículos. Sin embargo, durante el proceso de creación de la infraestructura han surgido una serie de imprevistos que se han resuelto pero que han supuesto disponer de una base de datos limitada para este trabajo. Algunos de estos imprevistos han sido la dificultad de excluir la sombra de los vehículos para obtener su tamaño, el cambio de ubicación del sensor y la forma de detectar la huella magnética de los vehículos.

Por otro lado, se ha observado que el sensor no se encuentra en el centro de la zona de paso de los vehículos puesto que, a pesar de estar colocado en el centro de la rampa, los vehículos tienden a pasar por el lateral debido a la ubicación de la rampa respecto al aparcamiento. Esto supone que parte de los vehículos que han pasado cuando el sistema estaba recogiendo datos no han quedado registrados porque no han pasado por la zona del sensor.

Para la realización de este trabajo se ha obtenido una base de datos de 160 vehículos en la que se guarda para cada vehículo la huella magnética, el video en el que pasa por la zona del sensor y la etiqueta con los datos característicos. En la figura 3.9 se muestra un ejemplo de los datos obtenidos.


```
{
    "vehiculo": [{
            "Direccion": "sale",
            "Timestamp inicio": 1572447343,
            "Velocidad (km/h)": 9.7,
            "ID video": "1572447343.mp4",
            "Tamagno (metros)": 3.89,
            "Timestamp fin": 1572447346
        }
    ]
}
```


(a) (b)



(c) (d) (e)

Figura 3.9: (a) Frame del video (b) etiqueta (c) Huella magnética en el eje X (d) Huella magnética en el eje Y (e) Huella magnética en el eje Z

4 Modelo de clasificación

A continuación, se detalla el proceso de creación del sistema de reconocimiento de los vehículos. El sistema está compuesto por un clasificador de vehículos según su dirección y dos estimadores de su tamaño y velocidad.

4.1 Introducción

Como se ha explicado en el capítulo 2, para desarrollar el sistema de reconocimiento de vehículo se decide utilizar redes neuronales debido a los buenos resultados obtenidos en los diferentes proyectos. Además, se decide utilizar para este sistema redes de tipo perceptrón o MLP por su capacidad para aprender relaciones no lineales.

Para obtener la dirección de los vehículos se implementa un clasificador binario, entra o sale del aparcamiento, y para obtener la velocidad y el tamaño, se implementan dos estimadores ya que, a diferencia de los trabajos mencionados en el capítulo 2 que clasifican los vehículos en grupos, el objetivo de este proyecto es obtener el tamaño exacto y su velocidad.

4.2 Tratamiento de los datos

Tras la recogida de datos se obtiene una base de datos de 160 huellas magnéticas con sus etiquetas formadas por las tres variables que se desea conocer de los vehículos que pasan por la zona del sensor. Estas variables son tamaño, velocidad y dirección. En la figura 4.1 se puede observar la distribución de los datos según su tamaño, velocidad y dirección, respectivamente. La base de datos contiene 81 vehículos que salen del aparcamiento y 79 que entran, con un tamaño medio de 4.27 metros y una velocidad media de 12.65 km/h.

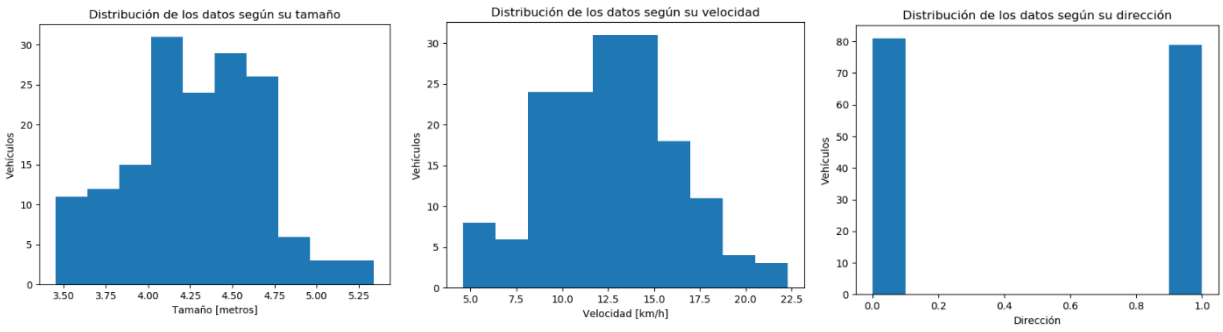


Figura 4.1: Distribución de los datos según su tamaño, velocidad y dirección

Para comprobar la cantidad de información que proporcionan la media y la desviación estándar de los datos respecto a las etiquetas se realiza la correlación cruzada. Para ello, se utiliza la librería Pandas [26] de Python, obteniendo los valores de correlación mostrados en tabla 4.1 y tabla 4.2. La media y la desviación estándar contienen algo de información, pero no la suficiente como para utilizarlas de *features* para las redes neuronales. Se estudia también si, teniendo en cuenta la dirección de los vehículos, la media y la desviación estándar aportan más información. Por un lado, se realiza la correlación cruzada de la media y la desviación estándar con las etiquetas de tamaño y velocidad según la dirección de los vehículos y, por el otro lado, se asigna valores positivos a la velocidad y el tamaño de los vehículos que entran al aparcamiento y negativos a los vehículos que salen y se realiza la correlación con las nuevas etiquetas. Los resultados de estas correlaciones, Anexo A.1, son similares a los anteriores.

Index	mean_x	mean_y	mean_z	len_vehicles	speed_vehicles	direction_vehicles
mean_x	1	0.153082	0.0568285	0.151749	0.122165	0.104746
mean_y	0.153082	1	0.0864338	-0.0666245	-0.116516	-0.0761586
mean_z	0.0568285	0.0864338	1	-0.0554197	-0.0351923	-0.0115536
len_vehicles	0.151749	-0.0666245	-0.0554197	1	-0.0439419	0.107536
speed_vehicles	0.122165	-0.116516	-0.0351923	-0.0439419	1	0.230441
direction_vehicles	0.104746	-0.0761586	-0.0115536	0.107536	0.230441	1

Tabla 4.1: Correlación entre las medias de los datos y las etiquetas

Index	std_x	std_y	std_z	len_vehicles	speed_vehicles	direction_vehicles
std_x	1	0.426589	0.448384	0.0520485	-0.12664	-0.167724
std_y	0.426589	1	0.603512	0.0573164	-0.121986	-0.0312932
std_z	0.448384	0.603512	1	0.199333	-0.223647	-0.139994
len_vehicles	0.0520485	0.0573164	0.199333	1	-0.0439419	0.107536
speed_vehicles	-0.12664	-0.121986	-0.223647	-0.0439419	1	0.230441
direction_vehicles	-0.167724	-0.0312932	-0.139994	0.107536	0.230441	1

Tabla 4.2: Correlación entre las desviaciones estándar de los datos y las etiquetas

Tras observar los coeficientes de correlación obtenidos se decide aplicar la normalización gaussiana a los datos. Para ello, se calcula la media y la desviación estándar de los tres ejes para cada vector de muestras y se normaliza utilizando la ecuación 1. Además, se remuestrean los datos de manera que todos tienen el mismo número de muestras.

$$\boldsymbol{X'} = \frac{\boldsymbol{X} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}, \quad \text{siendo } \mu \text{ la media de X y } \sigma \text{ su desviación estándar}$$

(Ecuación 1)

Tras normalizar los datos y remuestrearlos, se realiza una reducción de la dimensionalidad de los datos para visualizarlos en dos dimensiones utilizando Análisis de Componentes Principales (Principal Component Analysis, PCA). En la figura 4.2 se observan las proyecciones de los datos en dos dimensiones, utilizando como color de los puntos el valor de la etiqueta. En las proyecciones se observa que los datos de cada valor de la etiqueta no se sitúan en una distancia próxima, sino que todos los datos se encuentran dispersos, lo

que implica que no hay un patrón que la red neuronal pueda reconocer y utilizar de manera lineal. Sin embargo, en el caso de la etiqueta de dirección se aprecia una ligera agrupación de los datos en función del valor de la etiqueta, por lo que se repite el proceso, pero, esta vez, utilizando t-SNE (t-distributed stochastic neighbor embedding), figura 4.3. Con este método gran parte de los datos de cada valor de la etiqueta se sitúan cercanos, formándose dos grupos diferenciables. El código utilizado para el tratamiento de los datos se encuentra en el Anexo G.

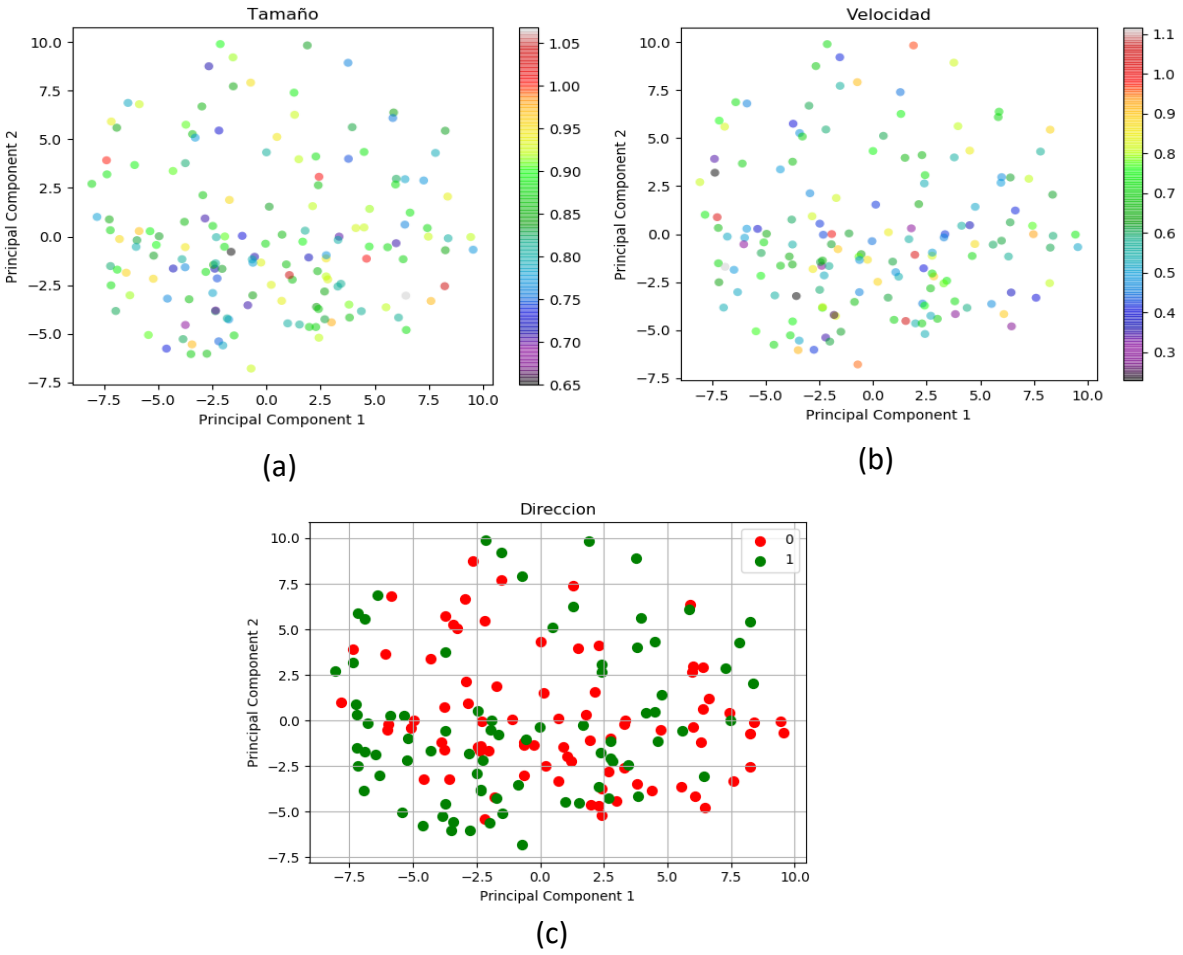


Figura 4.2: Proyección en 2D de los datos y etiquetas con PCA

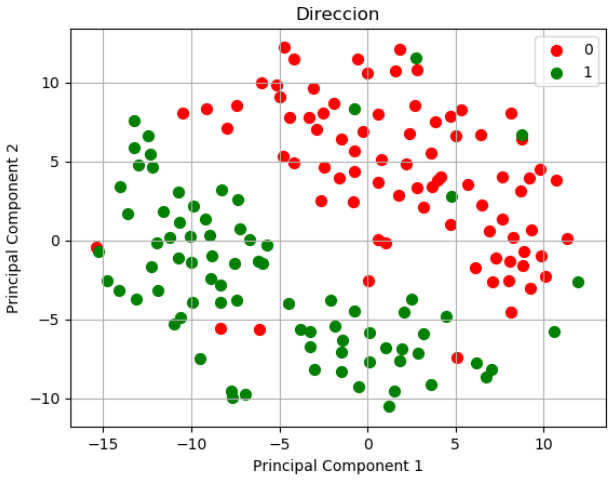


Figura 4.3: Proyección en 2D de los datos según la dirección con t-SNE

4.3 Selección de arquitectura

Como ya se ha comentado anteriormente, se decide implementar redes neuronales de tipo MLP, en concreto para obtener la dirección de los vehículos se implementa un clasificador binario en el que la salida puede ser 0, el vehículo sale del aparcamiento, o 1, el vehículo entra. Para obtener la velocidad y el tamaño, como se pretende estimar los valores reales, se implementan dos redes de regresión. Para utilizar redes de tipo MLP es importante la elección de la arquitectura neuronal, por lo que el siguiente paso es la elección de la arquitectura de cada una de las redes neuronales.

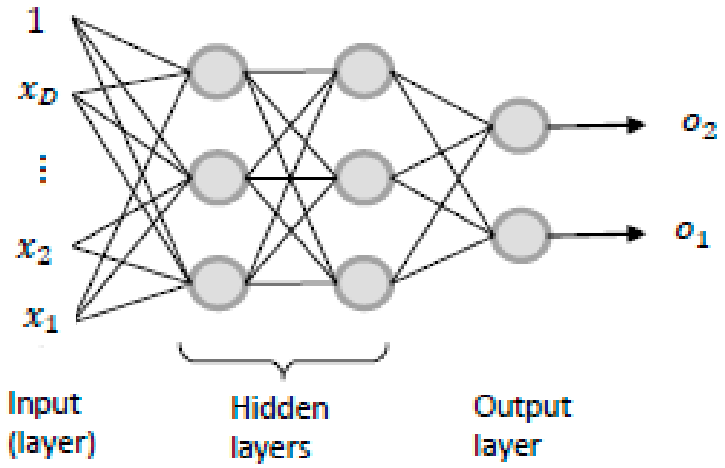


Figura 4.4: Arquitectura de un MLP

Un MLP está formado por la capa de entrada, la capa de salida y una o más capas intermedias, conocidas como capas ocultas, figura 4.4. Para la realización de este trabajo se decide diseñar los clasificadores con un máximo de dos capas ocultas. Como vector de entrada se utiliza las señales submuestreadas de los ejes concatenadas, seguido de la media y la desviación estándar de cada eje, ecuación 2. Por tanto, el número de datos de entrada de las redes depende del submuestreo aplicado. Por otro lado, para que los pesos sean más pequeños, se normalizan las etiquetas de tamaño y velocidad utilizando la ecuación 3:

$$input = [x, y, z, \mu_x, \sigma_x, \mu_y, \sigma_y, \mu_z, \sigma_z]$$

(Ecuación 2)

$$y' = \frac{y - y_{min}}{y_{max} - y_{min}}$$

(Ecuación 3)

Además, para comprobar la precisión de las redes se utiliza la validación cruzada, ya que se dispone de una base de datos pequeña. Para ello se distribuyen los datos en 10 grupos equilibrados, es decir, la mitad de los datos del grupo son de vehículos que entran al aparcamiento y la otra mitad de vehículos que salen.

Para obtener la dirección se diseña primero una red de tipo perceptrón, es decir, sin capa oculta. Se diseña una red en la que la entrada es el vector formado según la ecuación 2 y se implementa una sola neurona de salida con función de activación sigmoidea. El objetivo de la red es maximizar la precisión medida a través de la salida de la red y las etiquetas. Con este diseño se realizan distintos entrenamientos variando el submuestreo aplicado a los datos. Los resultados que se muestran en la tabla 4.3 son la media y la desviación estándar de los resultados obtenidos en la validación cruzada. Se observa que el mejor resultado, 75.62% ± 12.94% se ha obtenido con 20 muestras de cada eje, es decir, con un vector de entrada de 66 valores.

DIRECCION	Nº de muestras por eje				
	10	15	20	25	30
Precisión	74.37% ± 13.24%	73.12% ± 14.26%	75.62% ± 12.94%	73.75% ± 13.63%	73.75% ± 11.79%

Tabla 4.3: Resultados obtenidos variando el número de muestras por eje

A partir de estos resultados se diseña una red MLP para intentar mejorar la eficacia y se realizan dos conjuntos de simulaciones. En el primero, se diseña una red con una capa oculta con función de activación tangente hiperbólica y se realiza el método de validación cruzada variando el número de neuronas de la capa oculta. En la segunda, se diseña una red con dos capas ocultas con función de activación tangente hiperbólica y se varían el número de neuronas de las dos capas. Los resultados obtenidos en las simulaciones se encuentran en el Anexo B.1. El mejor resultado obtenido se consigue con la red de dos capas ocultas, cuando la primera capa oculta tiene 12 neuronas y la segunda tiene 5 neuronas. La eficacia conseguida con esta arquitectura es 78.75% ± 9.4%.

Sin embargo, dado el pequeño margen de mejora con esta arquitectura y teniendo en cuenta el aumento del coste computacional entre esta arquitectura y el perceptrón inicial, se decide escoger como arquitectura del clasificador de vehículos según la dirección un perceptrón con una neurona de salida y activación sigmoidea.

Por otro lado, para los estimadores de tamaño y velocidad se diseña inicialmente una red MLP con una capa oculta con función de activación tangente hiperbólica y una neurona de salida de tipo lineal. La función de coste que se desea minimizar en estas redes es el error cuadrático medio o MSE. Se obtiene además el valor del error porcentual absoluto medio o MAPE (Mean Absolute Percentage Error). Para obtener la arquitectura que mejores resultados proporciona se realiza un conjunto de simulaciones variando el número de neuronas de la capa oculta y el número de muestras de cada eje que se introducen a la red. Los resultados obtenidos del estimador del tamaño se encuentran en el Anexo B.2 y los resultados del estimador de la velocidad se encuentran en el Anexo B.3. En ambos casos los mejores resultados se obtienen cuando la capa oculta tiene 3 neuronas, obteniendo un MAPE de 20.96% ± 7.13% para el estimador del tamaño y 18.84% ± 2.78% para el estimador de la velocidad. Sin embargo, se observa que las redes no son capaces de encontrar características importantes de los datos, por lo que tienden a sacar el valor medio de los datos.

A continuación, se añade otra capa oculta para ver si en este caso, los resultados son mejores. Para ello, se realizan simulaciones variando el número de neuronas de las dos capas ocultas. Se decide introducir a las redes 20 muestras de cada eje, junto con las medias y las desviaciones estándar de cada eje. En los resultados mostrados en el Anexo A.2 y en el Anexo A.3, se observa que al aumentar las dimensiones de las redes no se produce una mejora en las estimaciones.

4.4 Resultados

Una vez escogida la arquitectura de las redes neuronales, se realiza el entrenamiento y la evaluación del sistema a partir del conjunto de test. Para ello, se entrena cada una de las redes durante 500 épocas con un factor de aprendizaje de 0.001 y, debido al tamaño de la base de datos, se utiliza el método de validación cruzada con 10 grupos equilibrados, como se ha explicado anteriormente.

4.4.1 Clasificador de los vehículos según la dirección

Para obtener la dirección se utiliza como función de coste la entropía cruzada binaria, ecuación 4, y se mide la precisión de la red a través de la salida de la red y las etiquetas. Tras realizar el método de validación cruzada de los datos, se obtiene una precisión de 74.37% ± 12.94%, siendo la media y la desviación estándar del conjunto de resultados obtenidos durante el método de validación cruzada. Para evaluar la red se divide la base de datos en un conjunto de train, 80% de los datos, y un conjunto de test, 20% de los datos. Se entrena el sistema con los datos de train y se evalúa con el conjunto de test. El código utilizado se muestra en el Anexo H. En la figura 4.5 se muestra un ejemplo de la evaluación de la red, en el cual se obtiene una precisión de 75% y la matriz de confusión mostrada en la tabla 4.4.

$$BCE = \begin{cases} -\log\big(f(s_1)\big) & si\ t_1 = 1 \\ -\log\big(1 - f(s_1)\big) & si\ t_1 = 0 \end{cases} \quad ,siendo\ s_1\ y\ t_1\ la\ salida\ predicha\ y\ la\ etiqueta$$

(Ecuación 4)

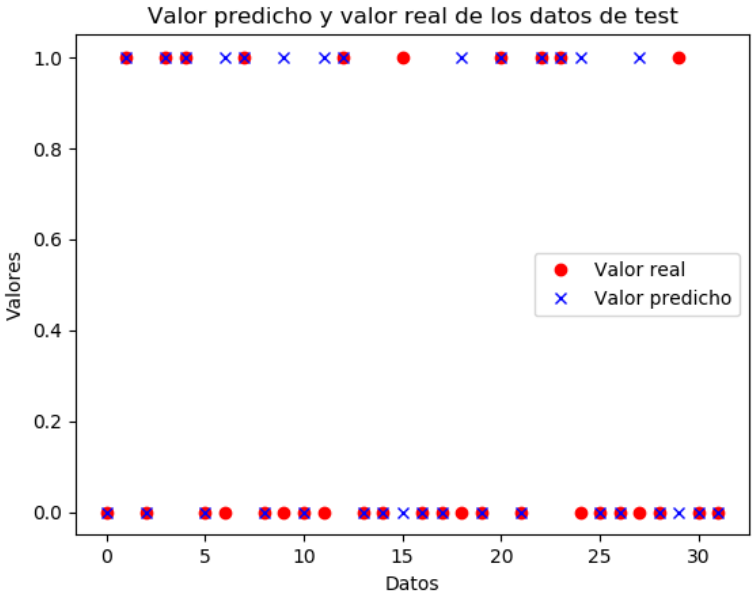


Figura 4.5: Valor real de la dirección y valor predicho por la red

	Clase real	
Clase asignada	Salida	Entrada
Salida	62	19
Entrada	23	56

Tabla 4.4: Matriz de confusión

Por último, dado que en la proyección en 2D de los datos utilizando t-SNE se observan los datos agrupados según la clase, se decide utilizar como entradas de la red las dos componentes principales. El método de proyección t-SNE no extrae las mismas componentes en cada ejecución, por lo que se realiza un conjunto de 10 simulaciones, obteniendo una precisión de 89.057% ± 1.01%

4.4.2 Estimador del tamaño de los vehículos

Para obtener el tamaño se utiliza como arquitectura final una red con una capa oculta con 3 neuronas y como función de coste se minimiza el error cuadrático medio o MSE, ecuación 5, y se mide el error porcentual absoluto medio o MAPE, ecuación 6. Tras realizar el método de validación cruzada de los datos, se obtiene 20.44% ± 5.26% de MAPE. El código implementado se encuentra en el Anexo I.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(t_i - s_i)^2 \quad siendo\ t_i\ y\ s_i\ el\ valor\ real\ y\ el\ valor\ estimado$$

(Ecuación 5)

$$MAPE = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{t_i - s_i}{t_i}\right| \quad siendo\ t_i\ y\ s_i\ el\ valor\ real\ y\ el\ valor\ estimado$$

(Ecuación 6)

Como se ha explicado anteriormente, la red no es capaz de estimar correctamente el tamaño, por lo que tiende a sacar como resultado la media de las etiquetas de entrenamiento, como se puede ver en la figura 4.6. En este ejemplo se puede observar cómo, a pesar de obtener 12.8% de MAPE, la red no es capaz estimar los valores.

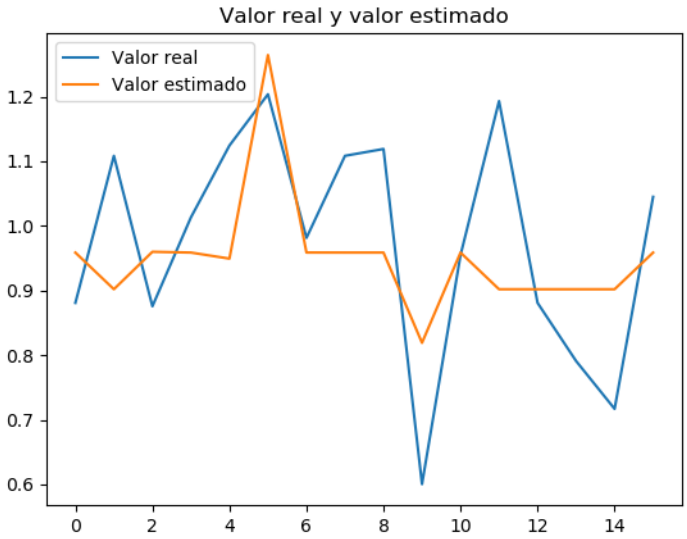


Figura 4.6: Valor real y valor estimado del tamaño en un subconjunto de validación

Además, durante la realización de las simulaciones se ha observado que la red no es capaz de sobreajustarse con los datos de entrenamiento, por lo que se puede concluir que la base de datos contiene información contradictoria o no proporciona suficiente información, por lo que la red asigna a la salida el valor medio de las etiquetas minimizando el error.

4.4.3 Estimador de la velocidad de los vehículos

En el caso de la estimación de la velocidad sucede lo mismo que en el estimador del tamaño. En esta ocasión se utiliza también una red con una capa oculta con 3 neuronas y la minimización del MSE como función de coste. Tras realizar el método de validación cruzada de los datos, se obtiene 21.54% ± 4.34% de MAPE. En la figura 4.7 se muestra un ejemplo de los valores estimados por la red frente a los valores reales habiendo obtenido 16.6% de MAPE. El código utilizado se encuentra en el Anexo J.

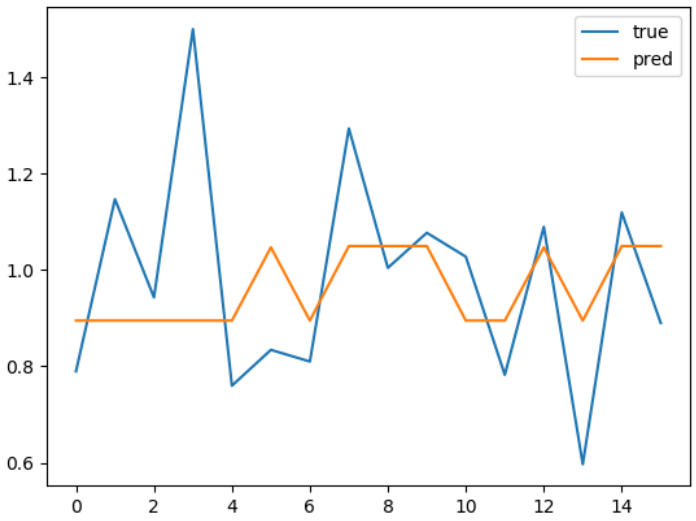


Figura 4.7: Valor real y valor estimado de la velocidad en un subconjunto de validación

5 Conclusiones y trabajo futuro

En este último capítulo se resumen la consecución de objetivos y el trabajo futuro que se puede realizar a partir de este Trabajo Fin de Máster.

5.1 Conclusiones

En este trabajo se ha estudiado la posibilidad de utilizar un sensor magnético para detectar los vehículos que pasan por la calzada, así como su tamaño, velocidad y dirección. Como resultado de este estudio se ha comprobado que, a partir de los datos obtenidos, es posible conocer la dirección de los vehículos, pero no se dispone de la suficiente información para estimar el tamaño y la velocidad. Esto podría deberse a la localización del sensor, ya que no se encuentra centrado en la zona de paso de los vehículos provocando variaciones en los datos recogidos, pero, tras el estudio realizado, se puede concluir que el principal motivo es que con un único magnetómetro no se obtiene la suficiente información del vehículo. Por tanto, se propone añadir un segundo magnetómetro al sensor, de manera que a partir de las señales obtenidas por los dos magnetómetros y la diferencia de tiempos de detección entre ambos se obtendría más información.

A continuación, se exponen los objetivos parciales de este trabajo y su grado de cumplimiento:

- Estudio sobre los métodos de clasificación: se ha realizado un análisis sobre los sistemas más utilizados actualmente para la monitorización del tráfico, así como las técnicas utilizadas para su clasificación. El análisis realizado se puede observar en el Capítulo 2.
- Desarrollo de la infraestructura para la creación de la base de datos: se ha desarrollado una infraestructura capaz de crear la base de datos automáticamente. Esto permite obtener una base de datos muy grande para entrenar los clasificadores, consiguiendo un sistema más robusto. Esta infraestructura se encuentra descrita en el Capítulo 3.
- Análisis de la base de datos: se ha realizado un análisis de los datos en el que se ha estudiado la correlación de los datos obtenidos por el sensor con las etiquetas asociadas. El análisis de los datos se puede observar en el Capítulo 4, apartado 2.
- Desarrollo de los clasificadores: se han desarrollado los clasificadores para obtener el tamaño, la velocidad y la dirección. Además, se ha aplicado un proceso de validación cruzada para estudiar la calidad de los clasificadores. Los clasificadores se encuentran en el Capítulo 4.

5.2 Trabajo futuro

A continuación, se destacan las posibles líneas de trabajo futuro que se pueden realizar a partir de los resultados obtenidos de este trabajo.

- Base de datos: Una línea de trabajo sería aumentar la base de datos utilizando la infraestructura desarrollada en este trabajo y repetir la fase de entrenamiento con una base de datos muy grande.

- Cambiar la ubicación del sensor y la cámara: Otra línea de trabajo sería colocar el sensor y la cámara en una ubicación en la que el sensor se encontrase centrado en la zona de paso de los vehículos. Además, sería conveniente que la nueva ubicación permitiese obtener datos de vehículos que circulasen a mayor velocidad.

- Añadir un segundo magnetómetro: Otro posible camino de mejora sería añadir al sensor un segundo magnetómetro. De manera que a partir de la diferencia de tiempo en detectar un cambio en el campo magnético permitiría obtener más información sobre el vehículo y mejorar, por tanto, la estimación de sus características. Este segundo magnetómetro permitiría también eliminar el ruido de fondo a partir de la diferencia entre las dos señales.

- Implementar los clasificadores en el sensor: como trabajo futuro sería implementar las clasificadores que se han diseñado en el sensor, de manera que el propio sensor obtendría el tamaño, la velocidad y la dirección del vehículo y únicamente enviaría esta información.

Bibliografía

[1] (2019). *HOWLab*. Available: http://howlab.unizar.es/.

[2] R. Calle, "Sistema De Identificación De Vehículos Basado En Campo Magnético.", Universidad de Zaragoza, 2018.

[3] S. Meta and M. G. Cinsdikici, "Vehicle-classification algorithm based on component analysis for single-loop inductive detector," *IEEE Transactions on Vehicular Technology,* vol. 59, *(6),* pp. 2795-2805, 2010.

[4] Y. Ki and D. Baik, "Vehicle-classification algorithm for single-loop detectors using neural networks," *IEEE Transactions on Vehicular Technology,* vol. 55, *(6),* pp. 1704-1711, 2006.

[5] Y. Lao *et al*, "Gaussian mixture model-based speed estimation and vehicle classification using single-loop measurements," *Journal of Intelligent Transportation Systems,* vol. 16, *(4),* pp. 184-196, 2012.

[6] J. Gajda and M. Stencel, "A highly selective vehicle classification utilizing dual-loop inductive detector," *Metrology and Measurement Systems,* vol. 21, *(3),* pp. 473-484, 2014.

[7] J. Sochor, A. Herout and J. Havel, "Boxcars: 3d boxes as cnn input for improved fine-grained vehicle recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 2016, pp. 3006-3015.

[8] A. Ghosh, M. Sabuj and H. H. Sonet, *A Real-Time Video-Based Adaptive Vehicle Counting, Speed Measurement and Classification Tool in Java,* 2019.

[9] S. Zhang *et al*, "A multi-features fusion method based on convolutional neural network for vehicle recognition," in *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI),* 2018, pp. 946-953.

[10] X. Le, Y. Wang and J. Jo, "Combining deep and handcrafted image features for vehicle classification in drone imagery," in *2018 Digital Image Computing: Techniques and Applications (DICTA),* 2018, pp. 1-6.

[11] E. C. Neto *et al*, "Brazilian vehicle identification using a new embedded plate recognition system," *Measurement,* vol. 70, pp. 36-46, 2015.

[12] L. N. Thu, A. Win and H. N. Oo, "Vehicle type classification based on acoustic signals using denoised MFCC," in *2018 IEEE International Conference on Information Communication and Signal Processing (ICICSP),* 2018, pp. 113-117.

[13] A. Mayvan, S. Beheshti and M. Masoom, "Classification of vehicles based on audio signals using quadratic discriminant analysis and high energy feature vectors," *International Journal on Soft Computing,* vol. 6, *(1),* pp. 53, 2015.

[14] M. Paulraj *et al*, "Moving vehicle recognition and classification based on time domain approach," *Procedia Engineering,* vol. 53, pp. 405-410, 2013.

[15] J. Nedoma *et al*, "Analysis of the use of fiber-optic sensors in the road traffic," *IFAC-PapersOnLine,* vol. 51, *(6),* pp. 420-425, 2018.

[16] J. Nedoma *et al*, "Non-destructive fiber-optic sensor system for the measurement of speed in road traffic," *Advances in Electrical and Electronic Engineering,* vol. 14, *(5),* pp. 602-608, 2016.

[17] M. Bin and Z. Xinguo, "Study of vehicle weight-in-motion system based on fiber-optic microbend sensor," in *2010 International Conference on Intelligent Computation Technology and Automation,* 2010, pp. 458-461.

[18] S. Vançin and E. Erdem, "Implementation of the vehicle recognition systems using wireless magnetic sensors," *Sādhanā,* vol. 42, *(6),* pp. 841-854, 2017.

[19] G. Burresi and R. Giorgi, "A field experience for a vehicle recognition system using magnetic sensors," in *2015 4th Mediterranean Conference on Embedded Computing (MECO),* 2015, pp. 178-181.

[20] X. Chen, *Road Vehicle Recognition and Classification using Magnetic Field Measurement,* 2018.

[21] Y. He, Y. Du and L. Sun, "Vehicle classification method based on single-point magnetic sensor," *Procedia-Social and Behavioral Sciences,* vol. 43, pp. 618-627, 2012.

[22] R. Bhattarya, *In-Node Machine Learning-Based Vehicle Classification.* California State University, Long Beach, 2018.

[23] *LoRaNet API*. Available: https://bitbucket.org/howlab/loranet-api/wiki/Home.

[24] *InfluxDB*. Available: https://www.influxdata.com/blog/getting-started-python-influxdb/.

[25] *OpenCV*. Available: https://docs.opencv.org/master/index.html.

[26] *Librería Pandas para Python*. Available: https://pandas.pydata.org/pandas-docs/stable/reference/index.html.

Anexos

Anexo A. Tratamiento de los datos

1. Correlaciones cruzadas de los datos y las etiquetas

I. Resultados de la correlación cruzada de las medias y las desviaciones estándar de los datos y las etiquetas de los vehículos que entran al aparcamiento:

Index	mean_x_entra	mean_y_entra	mean_z_entra	len_entra	speed_entra
mean_x_entra	1	0.120757	0.209025	0.156204	0.165599
mean_y_entra	0.120757	1	0.0504201	-0.103011	-0.0980167
mean_z_entra	0.209025	0.0504201	1	0.076104	-0.0198044
len_entra	0.156204	-0.103011	0.076104	1	-0.158613
speed_entra	0.165599	-0.0980167	-0.0198044	-0.158613	1

Tabla A.1: Correlación cruzada de las medias y etiquetas de vehículos que entran

Index	std_x_entra	std_y_entra	std_z_entra	len_entra	speed_entra
std_x_entra	1	0.49148	0.560138	0.111689	-0.100832
std_y_entra	0.49148	1	0.590434	0.117431	-0.0612526
std_z_entra	0.560138	0.590434	1	0.258719	-0.196215
len_entra	0.111689	0.117431	0.258719	1	-0.158613
speed_entra	-0.100832	-0.0612526	-0.196215	-0.158613	1

Tabla A.2: Correlación cruzada de las desviaciones estándar y etiquetas de vehículos que entran

II. Resultados de la correlación cruzada de las medias y las desviaciones estándar de los datos y las etiquetas de los vehículos que entran al aparcamiento:

Index	mean_x_sale	mean_y_sale	mean_z_sale	len_sale	speed_sale
mean_x_sale	1	0.197022	-0.0984282	0.129422	0.0378307
mean_y_sale	0.197022	1	0.124264	-0.0176281	-0.106921
mean_z_sale	-0.0984282	0.124264	1	-0.209755	-0.0507698
len_sale	0.129422	-0.0176281	-0.209755	1	0.0284539
speed_sale	0.0378307	-0.106921	-0.0507698	0.0284539	1

Tabla A.3: Correlación cruzada de las medias y etiquetas de vehículos que salen

Index	std_x_sale	std_y_sale	std_z_sale	len_sale	speed_sale
std_x_sale	1	0.391047	0.357844	0.0434548	-0.0880852
std_y_sale	0.391047	1	0.620941	0.00220899	-0.181673
std_z_sale	0.357844	0.620941	1	0.180475	-0.202996
len_sale	0.0434548	0.00220899	0.180475	1	0.0284539
speed_sale	-0.0880852	-0.181673	-0.202996	0.0284539	1

Tabla A.4: Correlación cruzada de las desviaciones estándar y etiquetas de vehículos que salen

III. Resultados de la correlación cruzada de las medias y las desviaciones estándar de los datos y las etiquetas tamaño y velocidad de los vehículos. Se ha modificado el signo de las etiquetas según la dirección de los vehículos.

Index	mean_x	mean_y	mean_z	len_vehicles	speed_vehicles	direction_vehicles
mean_x	1	0.153082	0.0568285	0.105096	0.117775	0.104746
mean_y	0.153082	1	0.0864338	-0.0794879	-0.0723874	-0.0761586
mean_z	0.0568285	0.0864338	1	0.000717789	-0.0081579	-0.0115536
len_vehicles	0.105096	-0.0794879	0.000717789	1	0.958789	0.99601
speed_vehicl…	0.117775	-0.0723874	-0.0081579	0.958789	1	0.964316
direction_ve…	0.104746	-0.0761586	-0.0115536	0.99601	0.964316	1

Tabla A.5: Correlación cruzada de las medias y las nuevas etiquetas de vehículos

Index	std_x	std_y	std_z	len_vehicles	speed_vehicles	direction_vehicles
std_x	1	0.426589	0.448384	-0.165077	-0.16066	-0.167724
std_y	0.426589	1	0.603512	-0.025914	-0.016044	-0.0312932
std_z	0.448384	0.603512	1	-0.136657	-0.134079	-0.139994
len_vehicles	-0.165077	-0.025914	-0.136657	1	0.958789	0.99601
speed_vehicl…	-0.16066	-0.016044	-0.134079	0.958789	1	0.964316
direction_ve…	-0.167724	-0.0312932	-0.139994	0.99601	0.964316	1

Tabla A.6: Correlación cruzada de las desviaciones estándar y las nuevas etiquetas de vehículos

Anexo B. Modelo de clasificación. Resultados

1. Clasificador de los vehículos según su dirección

1. Conjunto de simulaciones con una capa oculta, variando el número de neuronas, cuando la señal de entrada está compuesta por 20 muestras de cada eje y las medias y desviaciones estándar de cada eje. Los resultados obtenidos, formados por la media y la desviación estándar de los resultados obtenidos en la validación cruzada, se muestran en la tabla B.1.

Nº de neuronas en la capa oculta	Precisión
2	63.75% ± 12.9%
3	71.25% ± 14%
4	65% ± 10.1%
5	70.62% ± 8.9%
6	66.25% ± 11.2%
7	73.75% ± 13.9%
8	76.24 % ± 10%
9	76.62 % ± 11. 7%
10	67.5 % ± 11.4%
11	68.75 % ± 10.8%
12	68.12 % ± 10.2%
13	73.75 % ± 10.4%
14	75% ± 11.2%
15	71.25% ± 11.6%
16	73.12 % ± 7.9%
17	73.12 % ± 12.2%
18	73.12 % ± 9.7%
19	71.87 % ± 8%
20	72.5 % ± 10.9%
21	76.87 % ± 10.1%
22	76.87 % ± 12.2%
23	76.87 % ± 8.9%
24	73.75 % ± 10%
25	75 % ± 11.2%

Tabla B.1: Eficacia de la red de dirección variando el número de neuronas

2. Conjunto de simulaciones con dos capas ocultas. La señal de entrada está compuesta por 20 muestras de cada eje y las medias y desviaciones estándar de cada eje. Los resultados obtenidos se muestran en la tabla B.2.

Precisión	Nº neuronas en la 2º capa oculta								
Nº neuronas en la 1º capa oculta	2	3	4	5	6	7	8	9	10
2	61.25% ± 13.3%	60.62% ± 9.7%	61.25% ± 11.8%	64.37% ± 16.3%	57.5% ± 9.2%	69.37% ± 12.9%	63.75 % ± 10.4%	61.25 % ± 10.4%	65.62 % ± 14.3%
3	63.75% ± 11.8%	68.12% ± 9.9%	61.87% ± 9%	63.12% ± 12%	70% ± 16%	62.5% ± 12.8%	63.75% ± 12.1%	62.5% ± 17.2%	65% ± 9.8%
4	65.62% ± 12.3%	72.5% ± 12.6%	69.37% ± 12.9%	58.75% ± 10.1%	68.75% ± 8.8%	65% ± 11.9%	66.87% ± 14.3%	63.75% ± 12.1%	66.25% ± 7%
5	74.37% ± 13.8%	67.5% ± 13.9%	71.87% ± 14%	73.75% ± 9.6%	69.37% ± 10.6%	71.87% ± 9.4%	66.25% ± 13.5%	68.12% ± 15.4%	66.25% ± 13.2%
6	75.12% ± 12.8%	65% ± 10.1%	66.87% ± 7.4%	70.62% ± 8.9%	68.75% ± 7.9%	65.62% ± 14.8%	59.37% ± 12.9%	71.24% ± 17.3%	67.5% ± 12.4%
7	67.5% ± 6.7%	71.87% ± 12.6%	74.37% ± 9.4%	71.87% ± 9.4%	75.62% ± 9%	71.25% ± 8.5%	68.12% ± 13.8%	69.37% ± 12.6%	68.75% ± 13.7%
8	70.62% ± 12.82%	66.25% ± 11.25%	66.87% ± 20.55%	72.5% ± 10.53%	68.12% ± 11.33%	71.87% ± 11.94%	67.5% ± 13.34%	69.37% ± 11.33%	70% ± 16.95%
9	72.5% ± 11.6%	68.75% ± 15.1%	72.5% ± 14.3%	70.62% ± 9.3%	73.12% ± 7.9%	69.37% ± 9 %	68.12% ± 16.2%	71.25% ± 13.2%	70.62% ± 9.7%
10	75% ± 9.7%	70% ± 8.8%	73.75% ± 13.3%	68.75% ± 11.2%	63.75% ± 14.4%	69.37% ± 14.4%	73.12% ± 13.7%	66.87% ± 15.8%	71.87% ± 11.3%
11	71.25% ± 8.5%	71.24% ± 12.6%	73.12% ± 11.2%	71.25% ± 9.8%	73.12% ± 8.4%	70.62% ± 10.5%	76.24% ± 8.3%	68.75% ± 10.1%	75% ± 11.18%
12	73.75% ± 9.6%	71.25% ± 9.4%	70% ± 12.4%	78.75% ± 9.4%	72.5% ± 10.9%	71.87% ± 10.6%	75% ± 10.5%	74.37% ± 13.5%	64.37% ± 13.1%
13	72.5% ± 8.9%	71.25% ± 11.9%	70.62% ± 12.2%	76.25% ± 10.8%	68.75% ± 10.1%	70.62% ± 7.9%	70% ± 9.6%	75.62% ± 9%	72.5% ± 9.8%
14	73.75% ± 7.8%	71.87% ± 12.3%	75% ± 11.5%	68.75% ± 11.5%	71.25% ± 13.8%	75% ± 11.9%	72.5% ± 12.6%	71.87% ± 11.9%	70.62% ± 12.5%
15	77.5% ± 7.5%	75.62% ± 8.6%	69.37% ± 12.6%	76.87% ± 8.9%	71.25% ± 14.3%	70% ± 10%	71.87% ± 11.9%	68.12% ± 7.6%	68.12% ± 11.7%
16	68.75% ± 13.1%	71.25% ± 8%	72.5% ± 9.4%	74.37% ± 11.3%	66.87% ± 13.4%	73.75% ± 8.8%	72.5% ± 8%	70.62% ± 13.4%	72.5% ± 11.6%
17	73.12% ± 8.9%	70% ± 13.1%	71.25% ± 5.7%	71.25% ± 14%	75% ± 9.3%	74.37% ± 11%	71.25% ± 13.8%	73.75% ± 13.1%	71.25% ± 15.4%
18	73.12% ± 12.2%	71.25% ± 14.3%	69.37% ± 7.6%	73.12% ± 12.5%	76.87% ± 10.8%	69.37% ± 11.3%	72.5% ± 8%	71.87% ± 11.6%	73.75% ± 10.8%
19	70% ± 10.4%	65.62% ± 12.3%	70.62% ± 12.2%	75% ± 10.5%	69.37% ± 10.3%	75% ± 14%	76.24% ± 9.2%	65.62% ± 10.2%	69.37% ± 9%
20	73.75% ± 10%	73.75% ± 9.2%	72.5% ± 12.2%	73.75% ± 10%	75% ± 12.8%	73.12% ± 10.5%	69.4% ± 10.6%	74.37% ± 8.1%	73.12% ± 12.2%

Tabla B.2: Eficacia de la red de dirección variando el número de neuronas de las capas ocultas

3. Conjunto de simulaciones de la red utilizando como vector de entrada las dos componentes principales obtenidas en cada simulación utilizando t-SNE. Los resultados se muestran en la tabla B.3.

Simulación	Precisión
1	88.12% ± 7.09%
2	88.12% ± 9.86%
3	89.37% ± 8.86%
4	88.12% ± 8.12%
5	91.25% ± 9.35%
6	89.99% ± 10.15%
7	88.74% ± 7.8%
8	89.99% ± 8%
9	88.75% ± 9.6%
10	88.12% ± 9.45%

Tabla B.3: Eficacia de la red introduciendo las componentes principales de t-SNE

2. Estimador del tamaño de los vehículos

1. Conjunto de simulaciones con una capa oculta, variando el número de neuronas y el número de muestras de cada eje. Los resultados obtenidos, formados por la media y la desviación estándar de los resultados obtenidos en la validación cruzada, se muestran en la tabla B.4.

MAPE	Nº de muestras por eje				
Nº neuronas	10	15	20	25	30
3	21.41% ± 6.85%	21.12% ± 6.55%	20.96% ± 7.13%	21.8% ± 9.03%	20.8% ± 5.66%
4	22.86% ± 5.34%	21.05% ± 7.41%	21.41% ± 8.41%	23.09% ± 9.78%	23.49% ± 7.27%
5	22.1% ± 6.58%	22.42% ± 5.9%	21.64% ± 5.63%	22.07% ± 6.34%	22.43% ± 9.15%
6	22.06% ± 8.45%	22.12% ± 7.23%	24.33% ± 7.07%	26.07% ± 5.26%	25.98% ± 9.8%
7	22.27% ± 5.71%	23.59% ± 5.08%	21.76% ± 7.08%	22.8% ± 6.33%	27.2% ± 14.36%
8	21.24% ± 4.94%	23.51% ± 7.27%	25.01% ± 6.89%	23.13% ± 6.08%	26.94% ± 6.47%
9	21.74% ± 5.59%	21.93% ± 6.2%	25.44% ± 6.37%	25.03% ± 7.83%	28.11% ± 9.88%
10	24.31% ± 5.51%	23.7% ± 6.13%	26.02% ± 10.35%	25.63% ± 5.84%	27.53% ± 8.37%
11	23.14% ± 6.09%	26.78% ± 9.02%	24.88% ± 6.22%	27.33% ± 6.78%	23.14% ± 6.72%
12	24.66% ± 6.75%	25.87% ± 6.58%	25.98% ± 7.14%	25.51% ± 6.65%	27.27% ± 7.47%
13	24.75% ± 6.8%	26.42% ± 7.5%	26.49% ± 5.09%	28.35% ± 5.7%	25.85% ± 7.27%
14	25.07% ± 6.92%	26.09% ± 6.21%	25.39% ± 7.95%	28.04% ± 7.26%	26.91% ± 6.9%
15	28.07% ± 5.21%	25.4% ± 7.77%	26.26% ± 7.97%	27.66% ± 8.01%	26.73% ± 6.51%
16	25% ± 5.85%	25.34% ± 5.46%	26.08% ± 4.88%	30.08% ± 11.08%	26.63% ± 8.36%
17	26.19% ± 7.53%	30.41% ± 8.16%	27.58% ± 7.69%	28.21% ± 7.5%	29.31% ± 7.27%
18	27.11% ± 4.44%	31.54% ± 5.66%	32.63% ± 8.85%	32.07% ± 7.45%	31.81% ± 7.39%
19	24.15% ± 5.04%	29.07% ± 5.78%	31.01% ± 8.6%	30.87% ± 6.79%	30.73% ± 10.13%
20	24.53% ± 7%	30.11% ± 5.28%	33.41% ± 10.76%	30.46% ± 6.65%	31.28% ± 6.33%
21	27.54% ± 9.04%	27.23% ± 5.93%	33.24% ± 6%	30.94% ± 11%	35.33% ± 5.73%
22	28.03% ± 5.21%	32.76% ± 6.21%	31.43% ± 9.3%	35.31% ± 4.69%	29.74% ± 5.95%
23	28.68% ± 5.29%	33.56% ± 9.1%	33.07% ± 6.83%	32.18% ± 5.38%	35.04% ± 10.05%
24	28.68% ± 6.49%	29.88% ± 5.92%	31.66% ± 8.08%	31.39% ± 9.26%	35.22% ± 6.29%
25	28.67% ± 5.2%	29.32% ± 5.45%	27.39% ± 8.64%	32.99% ± 7.23%	31.24% ± 6.29%
26	31.17% ± 8.97%	30.59% ± 4.97%	30.91% ± 9.16%	30.41% ± 6.6%	36.8% ± 7.16%
27	31.03% ± 4.85%	30.62% ± 7.19%	32.13% ± 8.51%	32.12% ± 6.22%	34.59% ± 8.17%
28	32.45% ± 4.27%	32.97% ± 9.41%	31.4% ± 5.98%	31.17% ± 3.46%	34.64% ± 11.68%
29	30.59% ± 6.84%	30.68% ± 8.65%	30% ± 7.38%	35.04% ± 7.59%	37.34% ± 5.64%
30	33.16% ± 7.73%	29.04% ± 5.96%	32.75% ± 6.83%	34.52% ± 8.56%	33.75% ± 5.95%

Tabla B.4:MAPE obtenido variando el número de neuronas de la capa oculta y el vector de entrada

2. Conjunto de simulaciones con dos capas ocultas. La señal de entrada está compuesta por 20 muestras de cada eje y las medias y desviaciones estándar de cada eje. Los resultados obtenidos se muestran en la tabla B.5.

MAPE	Nº neuronas en la 2º capa oculta								
Nº neuronas en la 1º capa oculta	2	3	4	5	6	7	8	9	10
2	20.26% ± 6.6%	20.69% ± 6.2%	21.02% ± 7.64%	20.69% ± 6.8%	19.81% ± 6.18%	19.77% ± 6.1%	19.78% ± 5.8%	19.93% ± 6.6%	20.29% ± 5.6%
3	21.37% ± 5.7%	21.25% ± 7.5%	21.46% ± 7.7%	21% ± 8.2%	21.21% ± 6.7%	20.68% ± 7.7%	20.25% ± 6%	20.85% ± 7%	19.43% ± 5.8%
4	21.42% ± 6.3%	22.18% ± 5.4%	22.26% ± 7.2%	20.13% ± 4.9%	20.8% ± 7.6%	20.09% ± 6.4%	21.15% ± 5.9%	20% ± 6.1%	22.01% ± 7.2%
5	21.26% ± 5.4%	21.71% ± 5.6%	22.38% ± 6. 7%	21.74% ± 5.9%	22.07% ± 7.8%	21.81% ± 4.9%	22.23% ± 6.5%	21.8% ± 6.9%	20.59% ± 6.1%
6	20.55% ± 6.8%	21.08% ± 6. 7%	21.97% ± 6.5%	21.24% ± 7.2%	21.28% ± 5.9%	21.52% ± 7.8%	22.51% ± 5.2%	21.66% ± 6.9%	22.27% ± 5%
7	23.64% ± 7.5%	21.38% ± 7.6%	21.83% ± 5.4%	21.22% ± 5.6%	21.95% ± 6.6%	22.63% ± 5.3%	24.09% ± 5.8%	21.91% ± 7%	22.4% ± 5%
8	25.06% ± 7.9%	22.26% ± 8%	22.9% ± 6.8%	22.01% ± 6.9%	23.32% ± 6%	23.78% ± 6.1%	23.1% ± 5.8%	22.24% ± 5.8%	21.02% ± 7.6%
9	19.83% ± 5.45%	25.17% ± 5.8%	23.8% ± 8.8%	24.09% ± 8.8%	23.57% ± 7%	23.12% ± 6.7%	28.49% ± 9.6%	23.53% ± 6.4%	22.47% ± 6.7%
10	24.75% ± 8.2%	24.84% ±8.3 %	24.36% ± 4.2%	25.6% ± 9%	23.4% ± 5.8%	25.1% ± 8%	23.68% ± 6.5%	24.69% ± 7.8%	23.93% ± 5.7%
11	24.31% ± 6.9%	24.52% ± 5.8%	24.97% ± 10.4%	23.59% ± 6.4%	23.9% ± 5.8%	24.4% ± 7.4%	23.39% ± 7.64%	25.09% ± 6.5%	26.1% ± 5.1%
12	26.71% ± 5.6%	26.68% ± 7.1%	26.07% ± 7.4%	26.13% ± 6.9%	23.93% ± 7.2%	26.66% ± 8.5%	26% ± 5%	23.9% ± 8.1%	26.49% ± 8%
13	24.6% ± 6%	26.59% ± 9.6%	24.44% ± 5.1%	24.92% ± 6.5%	27.49% ± 6.9%	25.81% ± 7.3%	23.92% ± 8.3%	27.01% ± 7.4%	26.39% ± 5.8%
14	23.96% ± 6.6%	21.49% ± 5.4%	25.81% ± 5.9%	24.33% ± 4.6%	26.31% ± 7.8%	25.95% ± 6.1%	27.49% ± 6.2%	23.65% ± 6.7%	23.3% ± 6.1%
15	25.97% ± 7.7%	26.12% ± 6.44%	28.43% ± 7.8%	29.37% ± 7.6%	24.57% ± 7.9%	25.36% ± 5.5%	25.46% ± 6.4%	25.32% ± 7.9%	25.19% ± 6%
16	24.11% ± 5.6%	26.26% ± 6.6%	25.95% ± 5.2%	25.67% ± 5.7%	27.51% ± 8.2%	23.46% ± 6%	28.19% ± 5.8%	27.01% ± 7.9%	25.46% ± 4.1%
17	26.47% ± 6%	25.07% ± 6.2%	26.09% ± 7.6%	27.2% ± 8%	24.03% ± 7%	28.86% ± 7.55%	24.61% ± 4.9%	26.47% ± 7.1%	24.83% ± 7.1%
18	27.8% ± 6.8%	25.77% ± 6.3%	27.57% ± 7.2%	25.29% ± 5.6%	24.5% ± 7.8%	27.91% ± 5.7%	24.75% ± 7.6%	28.56% ± 6.1%	27.63% ± 6.38%
19	27.25% ± 8.5%	27.33% ± 9.4%	28.82% ± 6.5%	25.63% ± 6.2%	27.38% ± 6.3%	27.44% ± 5.6%	24.17% ± 4%	27.41% ± 5.3%	28.87% ± 8%
20	29.69% ± 9.4%	29.15% ± 9.1%	25.85% ± 9.3%	29.24% ± 5.2%	26.08% ± 6.1%	30.2% ± 8%	27.57% ± 7%	27.28% ± 7.2%	25.24% ± 5.7%

Tabla B.5: MAPE obtenido variando el número de neuronas de las capas ocultas

3. Estimador de la velocidad de los vehículos

1. Conjunto de simulaciones con una capa oculta, variando el número de neuronas y el número de muestras de cada eje. Los resultados obtenidos, formados por la media y la desviación estándar de los resultados obtenidos en la validación cruzada, se muestran en la tabla B.6.

MAPE	Nº de muestras por eje				
Nº neuronas	10	15	20	25	30
3	19.44% ± 5%	19.66% ± 2.89%	21.04% ± 3.92%	20.13% ± 3.45%	18.84% ± 2.78%
4	21.27% ± 4.96%	20.12% ± 4.93%	20.08% ± 3.46%	23.76% ± 9.28%	19.58% ± 3.31%
5	21.2% ± 5.03%	22.91% ± 8.47%	20.15% ± 5.88%	20.92% ± 5.11%	21.24% ± 6.04%
6	20.88% ± 3.47%	21.53% ± 2.97%	22.59% ± 3.49%	22.77% ± 4.76%	24.23% ± 7.37%
7	21.49% ± 4.69%	18.96% ± 3.32%	20.11% ± 3.31%	20.94% ± 4.95%	24.23% ± 8.6%
8	22.95% ± 4.15%	21.31% ± 6.07%	21.65% ± 4.79%	20.74% ± 2.36%	23.68% ± 4.77%
9	21.94% ± 3.18%	21.65% ± 3.52%	23.27% ± 5.31%	23.19% ± 5.86%	24.02% ± 6.16%
10	23.16% ± 5.54%	23.59% ± 5.4%	24.86% ± 6.03%	23.47% ± 5.55%	22.57% ± 4.27%
11	22.84% ± 3.46%	22.64% ± 3.67%	21.98% ± 4.88%	25.47% ± 5.34%	23.85% ± 4.39%
12	21.47% ± 4.33%	22.97% ± 5.65%	22.37% ± 4.5%	26.58% ± 8.78%	27.93% ± 6.78%
13	21.41% ± 3.08%	23.95% ± 4.97%	24.52% ± 8.11%	24.47% ± 4.25%	26.66% ± 5.21%
14	25.44% ± 3.24%	22.71% ± 6.42%	24.41% ± 5.11%	25.82% ± 8.19%	26.44% ± 5.06%
15	26.08% ± 6.19%	23.75% ± 2.98%	28.15% ± 5.91%	26.27% ± 5.15%	24.68% ± 3.79%
16	23.28% ± 5.72%	21.82% ± 3.42%	25.91% ± 4.11%	28.01% ± 5.34%	25.06% ± 5.87%
17	24.57% ± 5.3%	23.62% ± 4.92%	23.26% ± 4.37%	27.1% ± 5.55%	29.22% ± 8.08%
18	23.02% ± 4.55%	29.41% ± 7.6%	30.16% ± 7.56%	29.41% ± 4.59%	30.47% ± 9.45%
19	22.61% ± 2.75%	29.25% ± 5.85%	30.28% ± 4.85%	30.86% ± 8.02%	31.65% ± 9.41%
20	28% ± 8.43%	25.36% ± 9.14%	28.16% ± 5.01%	27.46% ± 6.47%	30.62% ± 5.24%
21	27.09% ± 6.17%	26.04% ± 4.93%	30.39% ± 5.04%	27.64% ± 4.25%	28.68% ± 4%
22	28.35% ± 4.63%	27.12% ± 7.25%	29.29% ± 7.45%	29.17% ± 5.86%	30.16% ± 8.73%
23	29.07% ± 6.33%	28.09% ± 5.5%	29.85% ± 5.45%	33.44% ± 4.48%	29.13% ± 4.66%
24	27.55% ± 8%	27.06% ± 5.34%	26.97% ± 3.62%	27.04% ± 5.76%	29.33% ± 4.72%
25	28.34% ± 5.55%	28.92% ± 6.58%	29.2% ± 4.32%	24.59% ± 6.14%	30.95% ± 6.49%
26	24.69% ± 5.41%	29.45% ± 4.75%	26.51% ± 6.07%	31.97% ± 7.72%	28.47% ± 6.08%
27	31.22% ± 6.67%	30.02% ± 5.59%	29.71% ± 8.18%	31.38% ± 7.02%	31.04% ± 4.76%
28	29.43% ± 5.9%	29% ± 7.8%	28.43% ± 6.66%	32.08% ± 8.62%	32.23% ± 8.56%
29	30.29% ± 7.73%	31.18% ± 6.4%	32.03% ± 4.29%	32.68% ± 6.13%	33.05% ± 7.86%
30	30.79% ± 4.81%	31.72% ± 5.18%	31.91% ± 7.94%	31.85% ± 6.57%	30.55% ± 5.22%

Tabla B.6: MAPE obtenido variando el número de neuronas de la capa oculta y el vector de entrada

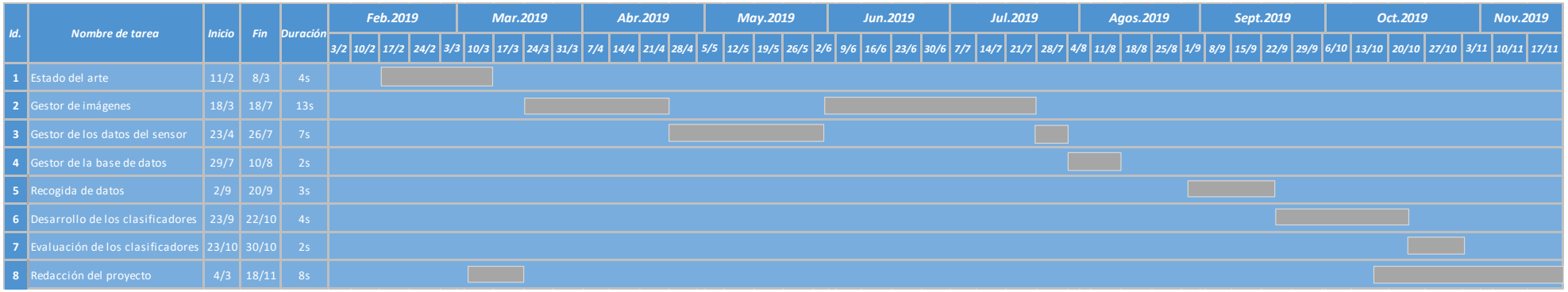
2. Conjunto de simulaciones con dos capas ocultas. La señal de entrada está compuesta por 20 muestras de cada eje y las medias y desviaciones estándar de cada eje. Los resultados obtenidos se muestran en la tabla B.7.

MAPE	Nº neuronas en la 2º capa oculta								
Nº neuronas en la 1º capa oculta	2	3	4	5	6	7	8	9	10
2	19.9% ± 4.9%	19.03% ± 3.1%	19.49% ± 3.2%	19% ± 3.1%	18.63% ± 2.9%	18.3% ± 3%	19.46% ± 3.3%	19.3% ± 2.9%	18.81% ± 3.5%
3	20.09% ± 3.1%	19.48% ± 4.1%	18.82% ± 3%	19.06% ± 3%	19.7% ± 4.4%	19.39% ± 3.7%	19.73% ± 4%	19.65% ± 3.2%	19.01% ± 2.4%
4	19.99% ± 4.6%	19.72% ± 2.8%	20.61% ± 4.7%	20.65% ± 3.1%	18.95% ± 3.67%	22.86% ± 5.2%	19.39% ± 3.3%	18.81% ± 4.1%	18.32% ± 3.6%
5	20.07% ± 2.4%	20.4% ± 3.8%	18.87% ± 3.1%	20.03% ± 3.8%	20.61% ± 3.73%	20.79% ± 2.45%	20.32% ± 3.44%	19.2% ± 3.32%	19.63% ± 2.9%
6	18.53% ± 3.3%	18.76% ± 3.17%	19.4% ± 3.8%	21.3% ± 5.6%	19.9% ± 3.1%	19.05% ± 3.1%	21.7% ± 3.8%	20.04% ± 3%	19.94% ± 3.8%
7	20.61% ± 4.2%	20.85% ± 2.2%	20.51% ± 4.1%	20.42% ± 4.4%	19.28% ± 3%	20.08% ± 3.9%	22.49% ± 4.2%	20.49% ± 5%	21.18% ± 3.5%
8	21.38% ± 3.3%	22.66% ± 4.5%	23.1% ± 3.7%	21.4% ± 3.9%	21.74% ± 4.7%	21.03% ± 4.2%	21.9% ± 4%	20.83% ± 4.2%	20.22% ± 4.2%
9	21.55% ± 4%	19.61% ± 4.3%	23.02% ± 4.3%	21.47% ± 3.3%	21.36% ± 3.4%	25.01% ± 6.3%	19.95% ± 2.7%	21.23% ± 5.2%	20.66% ± 4.1%
10	21.12% ± 3.6%	20.45% ± 4.7%	25.49% ± 4.25%	23.94% ± 2.7%	21.66% ± 3.2%	22.22% ± 3.9%	22.52% ± 4.9%	23.74% ± 5.83%	21.94% ± 5.6%
11	21.58% ± 3%	21.99% ± 5.4%	23.88% ± 6.49%	23.38% ± 4.6%	23.65% ± 4.3%	21.14% ± 3.9%	21.56% ± 5.3%	20.57% ± 4.3%	21.8% ± 3.2%
12	21.51% ± 4.4%	22.36% ± 5.6%	26.03% ± 6.4%	25.15% ± 4.9%	22.27% ± 4.3%	23.82% ± 3.9%	25.92% ± 4%	23.15% ± 3.5%	24.99% ± 3.3%
13	24.6% ± 3.5%	23.57% ± 4.7%	23.37% ± 4.6%	22.04% ± 4.1%	22.02% ± 3.25%	22.05% ± 3.7%	23.94% ± 4.3%	22.08% ± 3.8%	23.69% ± 4.5%
14	25.03% ± 5.1%	24.64% ± 7.4%	23.88% ± 3.5%	25.24% ± 5%	26.12% ± 6.4%	25.21% ± 5.9%	27.21% ± 5.9%	21.88% ± 3.9%	22.19% ± 4.2%
15	23.15% ± 7.2%	24.48% ± 6.8%	22.14% ± 4.3%	26.13% ± 4.6%	23.67% ± 6.1%	24.41% ± 6.4%	22.88% ± 4.2%	21.92% ± 4.1%	24.19% ± 3.8%
16	24.66% ± 4.9%	23.45% ± 4%	22.93% ± 6.3%	22.84% ± 6.9%	24.74% ± 4.4%	22.23% ± 5.3%	23.45% ± 4.5%	25.09% ± 6.3%	22.61% ± 4.5%
17	25.23% ± 4.9%	23.57% ± 5.5%	26.32% ± 8.4%	25.02% ± 6.4%	24.58% ± 4%	23.74% ± 5.9%	24.02% ± 4.3%	24.23% ± 4.1%	26.15% ± 7.4%
18	23.27% ± 5.5%	24.59% ± 3.2%	22.47% ± 3.1%	26.58% ± 4.7%	25.22% ± 3.6%	24.9% ± 5%	24.53% ± 4.5%	23.8% ± 4.9%	26.63% ± 5.4%
19	28.08% ± 5.5%	26.34% ± 4.3%	26.69% ± 4.6%	26.87% ± 4.3%	25.41% ± 3.8%	25.5% ± 5.9%	26.06% ± 4.3%	22.4% ± 5.3%	23.19% ± 3.6%
20	27.3% ± 4.5%	27.28% ± 4.8%	26.09% ± 7.6%	26.67% ± 4.3%	26.21% ± 5.7%	24.67% ± 3.6%	23.48% ± 5.7%	27.76% ± 5.5%	23.72% ± 4.3%

Tabla B.7: MAPE obtenido variando el número de neuronas de las capas ocultas

Anexo C. Cronograma de la planificación temporal

Id.	Nombre de tarea	Inicio	Fin	Duración
1	Estado del arte	11/2	8/3	4s
2	Gestor de imágenes	18/3	18/7	13s
3	Gestor de los datos del sensor	23/4	26/7	7s
4	Gestor de la base de datos	29/7	10/8	2s
5	Recogida de datos	2/9	20/9	3s
6	Desarrollo de los clasificadores	23/9	22/10	4s
7	Evaluación de los clasificadores	23/10	30/10	2s
8	Redacción del proyecto	4/3	18/11	8s



Anexo D. Código del gestor de imágenes

```
import imutils
import cv2 as cv
import numpy as np
import json
import time
import datetime
import os
from moviepy.video.io.ffmpeg_tools import ffmpeg_extract_subclip
from ftplib import FTP


def timestamp(frame_date):
    digits = {}

    # Lee las imagenes con los digitos de referencia
    cero = cv.imread('0.png')
    cero = cv.cvtColor(cero, cv.COLOR_BGR2GRAY)
    uno = cv.imread('1.png')
    uno = cv.cvtColor(uno, cv.COLOR_BGR2GRAY)
    dos = cv.imread('2.png')
    dos = cv.cvtColor(dos, cv.COLOR_BGR2GRAY)
    tres = cv.imread('3.png')
    tres = cv.cvtColor(tres, cv.COLOR_BGR2GRAY)
    cuatro = cv.imread('4.png')
    cuatro = cv.cvtColor(cuatro, cv.COLOR_BGR2GRAY)
    cinco = cv.imread('5.png')
    cinco = cv.cvtColor(cinco, cv.COLOR_BGR2GRAY)
    seis = cv.imread('6.png')
    seis = cv.cvtColor(seis, cv.COLOR_BGR2GRAY)
    siete = cv.imread('7.png')
    siete = cv.cvtColor(siete, cv.COLOR_BGR2GRAY)
    ocho = cv.imread('8.png')
    ocho = cv.cvtColor(ocho, cv.COLOR_BGR2GRAY)
    nueve = cv.imread('9.png')
    nueve = cv.cvtColor(nueve, cv.COLOR_BGR2GRAY)
    guion = cv.imread('-.png')
    guion = cv.cvtColor(guion, cv.COLOR_BGR2GRAY)
    puntos = cv.imread('dos puntos.png')
    puntos = cv.cvtColor(puntos, cv.COLOR_BGR2GRAY)

    digits[0] = cv.resize(cero, (60, 90))
    digits[1] = cv.resize(uno, (60, 90))
    digits[2] = cv.resize(dos, (60, 90))
    digits[3] = cv.resize(tres, (60, 90))
    digits[4] = cv.resize(cuatro, (60, 90))
    digits[5] = cv.resize(cinco, (60, 90))
    digits[6] = cv.resize(seis, (60, 90))
    digits[7] = cv.resize(siete, (60, 90))
    digits[8] = cv.resize(ocho, (60, 90))
    digits[9] = cv.resize(nueve, (60, 90))
    digits[10] = cv.resize(guion, (60, 90))
    digits[11] = cv.resize(puntos, (60, 90))

    frame_date = cv.cvtColor(frame_date, cv.COLOR_BGR2GRAY)
    frame_date = cv.threshold(frame_date, 50, 255, cv.THRESH_BINARY)[1]

    output = ''
    groupOutput = []
    w = 24

    for i in range(19):
        # Busca el digito de referencia que mas se parece a cada digito de la fecha
        x0 = 4 + i * w
        x1 = 5 + (i+1) * w
        roi = frame_date[:,x0:x1]
        roi = cv.resize(roi, (60, 90))
        scores = []

        for (digit, digitROI) in digits.items():
```

```
            result = cv.matchTemplate(roi, digitROI, cv.TM_CCOEFF_NORMED)
            (_, score, _, _) = cv.minMaxLoc(result)
            scores.append(score)

        if np.argmax(scores) == 10:
            groupOutput.append('-')
        elif np.argmax(scores) == 11:
            groupOutput.append(':')
        else:
            groupOutput.append(str(np.argmax(scores)))

    for i in range(len(groupOutput)):
        if not i == 10:
            output = output + groupOutput[i]
        else:
            output = output + ' '

    return output
    cv.waitKey(20)


class Vehicle:
    def __init__(self, t_start, x, size):
        self.t_start = t_start
        self.t_stop = ' '
        self.size = size
        self.speed = 0
        self.direction = ' '
        self.video = cv.VideoWriter
        self.num_frames = 1
        self.cont_speed = 0
        self.x = x
        self.w = 0
        self.window = 1 # 1: está dentro de la ventana, 0: no esta


# Conexion con el servidor FTP
ftp = FTP()
ftp.set_debuglevel(1)
ftp.connect('howlab.i3a.es', 21)
ftp.login('howlab', 'ppito56')
ftp.cwd('/')

# Inicializa variables
found = 0 # 1: vehiculo registrado, 0: vehiculo nuevo
area_min = 20000
thr = 40
meters = 8
h0_window = 50
h1_window = 370
w0_window = 310
w1_window = 1170
x0 = 400 # extremo izquierdo de la ventana de la zona sensor
x1 = 620 # extremo derecho de la ventana de la zona sensor
w_window = w1_window - w0_window
h_window = h1_window - h0_window

try:
    while 1:
        # Obtiene una lista con los videos que hay en el servidor sin procesar
        files = ftp.nlst('/videos')
        if not len(files) == 0:
            vehicles = []
            label = {}
            date = ' '
            num_frame = 0
            delete = 0

            # Descarga un video del servidor
            local_path = os.path.join(os.getcwd(), 'video.mp4')
            file = open(local_path, 'wb')
            ftp.retrbinary('RETR ' + files[0], file.write)
            file.close()
```

```
# Carga el video
cap = cv.VideoCapture('video.mp4')

# Lee el primeer frame
ret, old_f = cap.read()

while old_f is None:
   #Video dañado -> Borra el video del servidor y lee el siguiente
   ftp.delete(files[0])

   files = ftp.nlst('/videos')
   if not len(files) == 0:
      local_path = os.path.join(os.getcwd(), 'video.mp4')
      file = open(local_path, 'wb')
      ftp.retrbinary('RETR ' + files[0], file.write)
      file.close()

      cap = cv.VideoCapture('video.mp4')
      ret, old_f = cap.read()

# Transforma la imagen a escala de grises
old_frame = old_f[h0_window:h1_window, w0_window:w1_window]
old_gray = cv.cvtColor(old_frame, cv.COLOR_BGR2GRAY)
old_gray = cv.GaussianBlur(old_gray, (11, 11), 0)

# El primer frame será la imagen de fondo que se va a restar
firstFrame = old_gray
frame_date = old_f[10:45, 1450:1915]
date_init = timestamp(frame_date)
try:
   date_seg = time.mktime(datetime.datetime.strptime(date_init,
                                                     "%Y-%m-%d %H:%M:%S").timetuple())
   date_seg_init = int(date_seg)
except:
   print('Error al reconocer fecha')

# Flujo optico
# parametros para detección de esquinas ShiTomasi
feature_params = dict( maxCorners = 5,
                       qualityLevel = 0.3,
                       minDistance = 10,
                       blockSize = 10)
# Parámetros para el flujo óptico de Lucas Kanade
lk_params = dict( winSize = (15,15),
                  maxLevel = 2,
                  criteria = (cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 0.05))
mask = None
p0 = cv.goodFeaturesToTrack(old_gray, mask = mask, **feature_params)

while True:
   # Lee el siguiente frame
   ret, f = cap.read()

   if f is None:
      # Final del video
      Break

   num_frame = num_frame + 1

   frame_date = f[10:45, 1450:1915]

   # Transforma la imagen a escala de grises
   frame = f[h0_window:h1_window,w0_window:w1_window]
   gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
   gray = cv.GaussianBlur(gray, (11, 11), 0)

   # Resta el fondo al frame actual y aplica un umbral
   frameDelta = cv.absdiff(firstFrame, gray)
   thresh = cv.threshold(frameDelta, thr, 255, cv.THRESH_BINARY)[1]

   # Encuentra los blobs en la imagen
   thresh = cv.dilate(thresh, None, iterations=4)
   blobs = cv.findContours(thresh.copy(), cv.RETR_EXTERNAL,
                           cv.CHAIN_APPROX_SIMPLE)
```

```
blobs = imutils.grab_contours(blobs)

# Flujo optico
p1, st, err = cv.calcOpticalFlowPyrLK(old_gray, gray, p0, None, **lk_params)
good_new = p1[st==1]
good_old = p0[st==1]

# Analiza cada blob
for c in range(len(blobs)):
    (x, y, w, h) = cv.boundingRect(blobs[c])

    # Si el video empieza con un blob en medio de la zona del sensor -> borra video
    if num_frame == 1 and x <= x1 and x >= x0:
        delete = 1
        break

    area = cv.contourArea(blobs[c])

    # Si el blob es pequeño, no es un vehiculo
    if area < area_min:
        continue

    found = 0
    date_seg = 0

    for vehicle in vehicles:
        if vehicle.window:
            if x < x1 and vehicle.x < x + 60 and (vehicle.x >= x or vehicle.x == 0):
                # El vehiculo sale
                if x+w <= x0 and not vehicle.speed == 0:
                    vehicle.window = 0
                    vehicle.direction = 'sale'
                    vehicle.size = vehicle.size / vehicle.num_frames # tamaño medio
                    vehicle.size = round(vehicle.size, 2)
                    vehicle.speed = vehicle.speed / vehicle.cont_speed
                    vehicle.speed = round(vehicle.speed, 2)
                    date = timestamp(frame_date)
                    print(date)
                    try:
                        date_seg = time.mktime(datetime.datetime.strptime(date,
                                                          "%Y-%m-%d %H:%M:%S").timetuple())
                    except:
                        print('Error al reconocer fecha')
                    vehicle.t_stop = int(date_seg+1)
                if x > x0-200:
                    vehicle.num_frames = vehicle.num_frames + 1
                    size = w * meters/w_window # metros
                    vehicle.size = vehicle.size + size
                    for i,(new,old) in enumerate(zip(good_new,good_old)):
                        a,b = new.ravel()
                        c,d = old.ravel()
                        if a > x and a < (x+w) and b > y and b < (y+h):
                            vehicle.cont_speed = vehicle.cont_speed + 1
                            #(Diferencia de pixeles)*(25 fps) = pixeles por segundo
                            speed = np.abs(a-c)*25
                            speed = speed * meters/w_window # metros
                            speed = speed * 3.6 # km/h
                            vehicle.speed = vehicle.speed + speed
                            break
                    vehicle.x = x
                    found = 1
                    cv.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
                    break
                elif x+w >= x0-100 and vehicle.x <x and vehicle.x >x-60 and not vehicle.x==0:
                    # El vehículo entra
                    if x >= x1-40 and not vehicle.speed == 0:
                        vehicle.window = 0
                        vehicle.direction = 'entra'
                        vehicle.size = vehicle.size / vehicle.num_frames # tamaño medio
                        vehicle.size = round(vehicle.size, 2)
                        vehicle.speed = vehicle.speed / vehicle.cont_speed
                        vehicle.speed = round(vehicle.speed, 2)
                        date = timestamp(frame_date)
                        print(date)
```

```
                        try:
                            date_seg = time.mktime(datetime.datetime.strptime(date,
                                                                              "%Y-%m-%d %H:%M:%S").timetuple())
                        except:
                            print('Error al reconocer fecha')
                        vehicle.t_stop = int(date_seg)
                    if x+vehicle.w <= x1 + 100:
                        vehicle.num_frames = vehicle.num_frames + 1
                        vehicle.w = w
                        size = w * meters/w_window # metros
                        vehicle.size = vehicle.size + size
                        for i,(new,old) in enumerate(zip(good_new,good_old)):
                            a,b = new.ravel()
                            c,d = old.ravel()
                            if a > x and a < (x+w) and b > y and b < (y+h):
                                vehicle.cont_speed = vehicle.cont_speed + 1
                                #(Diferencia de pixeles)*(25 fps) = pixeles por segundo
                                speed = np.abs(a-c)*25
                                speed = speed * meters/w_window # metros
                                speed = speed * 3.6 # km/h
                                vehicle.speed = vehicle.speed + speed
                                break
                    vehicle.x = x
                    found = 1
                    cv.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
                    break

            if found or x+w < (x0+60) or x > x1:
                continue

            date = timestamp(frame_date)
            print(date)
            try:
                date_seg = time.mktime(datetime.datetime.strptime(date,
                                                                  "%Y-%m-%d %H:%M%S").timetuple())
            except:
                print('Error al reconocer fecha')

            vehicles.append(Vehicle(int(date_seg),x,w*meters/w_window))

        if delete:
            # Video no valido -> borra video
            Break

        # Crea la mascara del flujo optico
        mask = np.zeros_like(gray)
        mask[:] = 255
        for x, y in [np.int32(tr[-1]) for tr in p0]:
            cv.circle(mask, (x, y), 5, 0, -1)
            cv.rectangle(mask, (0, 0), (600, 60), 0, -1)
            cv.rectangle(mask, (0, 190), (600, 250), 0, -1)

        p = cv.goodFeaturesToTrack(gray, mask = mask, **feature_params)

        # Actualiza los puntos anteriores
        if p is not None:
            p0 = p
        else:
            p0 = p1

        old_gray = gray.copy()

        k = cv.waitKey(20) & 0xff
        if k == 27:
            break

    # Final del video
    cap.release()
    cv.destroyAllWindows()

    # Crea las etiquetas
    if len(vehicles) == 0:
        print('------No hay vehiculos')
    else:
```

```
                for vehicle in vehicles:
                    if vehicle.direction == 'sale' or vehicle.direction == 'entra':
                        path_video = '/base_datos/videos/'
                        path_label = '/base_datos/labels/'

                        # Corta el video y lo guarda con el timestamp como nombre
                        start = vehicle.t_start - date_seg_init - 1
                        stop = start + (vehicle.t_stop - vehicle.t_start) + 3
                        ffmpeg_extract_subclip('video.mp4', start, stop,
                                               targetname=str(vehicle. t_start)+".mp4")

                        label['vehiculo'] = []
                        label['vehiculo'].append({
                                               'ID video': str(vehicle.t_start)+".mp4",
                                               'Timestamp inicio': vehicle.t_start,
                                               'Timestamp fin': vehicle.t_stop,
                                               'Tamagno (metros)': vehicle.size,
                                               'Velocidad (km/h)': vehicle.speed,
                                               'Direccion': vehicle.direction
                        })

                        with open(str(vehicle.t_start) + '.json', 'w') as outfile:
                            json.dump(label, outfile)

                        #Manda el fichero y el video al servidor
                        fp = open(str(vehicle.t_start) + '.json', 'rb')
                        filename = path_label + str(vehicle.t_start) + '.json'
                        ftp.storbinary('STOR %s' % filename, fp, 1024 )
                        fp.close()

                        fp = open(str(vehicle.t_start)+".mp4", 'rb')
                        filename = path_video + str(vehicle.t_start) + '.mp4'
                        ftp.storbinary('STOR %s' % filename, fp, 1024 )
                        fp.close()

                        cv.waitKey(20)

                        # Borra el fichero y el video de la maquina local
                        os.remove(str(vehicle.t_start) + '.json')
                        os.remove(str(vehicle.t_start) + '.mp4')

            # Borra el video del servidor
            ftp.delete(files[0])

        # Cierra conexion con servidor ftp
        ftp.quit()

    except KeyboardInterrupt:
        print('Gestor parado')
        ftp.quit()
        cap.release()
        cv.destroyAllWindows()
```

Anexo E. Código del gestor de datos del sensor

```
import json
import serial
import time
import socket
import threading
import struct
from influxdb import InfluxDBClient

def socket_server_recv():
    # Socket creado para recibir las peticiones LORA del cliente
    global data
    global new_data
    global msg
    global new_msg
    global connection

    try:
        while 1:
            if stop_threads:
                break
            connection = None
            # Espera una conexion
            connection, client_address = sock.accept()
            while 1:
                if stop_threads:
                    break
                d = connection.recv(256)
                if d == b'':
                    print('El cliente se ha ido. Se cierra la conexion establecida')
                    break
                if not d == b'\r\n':
                    new_data = 1
                    data = d
                time.sleep(0.01)
    except:
        m = 'Se ha producido un error. Vuelva a intentarlo. \r\n'
        msg = bytes(m,'utf-8')
        new_msg = 1
        time.sleep(0.1)


def socket_server_send():
    # Socket creado para mandar al cliente las respuestas a las peticiones
    global new_resp
    global new_msg
    global msg

    try:
        while 1:
            if stop_threads:
                break
            while not connection == None:
                if stop_threads:
                    break
                if new_resp and (getP or setP or establish or destroy or update or send
                                 or beacon or broadcast):
                    new_resp = 0
                    connection.sendall(resp)
                elif new_msg and not getP and not setP and not establish and not destroy and
                                 not update and not send and not beacon and not broadcast:
                    new_msg = 0
                    connection.sendall(msg)
                time.sleep(0.01)
    except:
        m = 'Se ha producido un error. Vuelva a intentarlo. \r\n'
        msg = bytes(m,'utf-8')
        new_msg = 1
        time.sleep(0.1)
```

```
def com_serial():
    # Se encarga de la comunicacion por el puerto serie
    global msg
    global new_msg
    global send_data
    global new_beacon

    try:
        while 1:
            # Lee todo lo que hay en el buffer de entrada
            if stop_threads:
                break
            msg = p_serie.read(p_serie.inWaiting())
            if 'BEACON' in str(msg):
                new_beacon = 1
            elif not msg == b'':
                if 'ERROR' in str(msg) and not 'CRCERROR' in str(msg):
                    e = str(msg).split(':')[1]
                    e = e.split('\\r')[0]
                    des = getErrorByTextname(e)
                    e = "ERROR:{!s} - {!s}".format(e, des)
                    msg = bytes(e, 'utf-8')
                new_msg = 1

            if send_data:
                send_data = 0
                p_serie.write(data)
                time.sleep(0.01)

            time.sleep(1)

    except:
        m = 'Se ha producido un error. Vuelva a intentarlo. \r\n'
        msg = bytes(m,'utf-8')
        new_msg = 1
        time.sleep(0.01)


def getCommandByName(name):
    for command in commands:
        if command["name"] == name:
            return command['textname'] + '\r\n'
            break

    # Comando no encontrado
    raise Exception("Method with name '%s' not found" % name)


def getErrorByTextname(name):
    for error in commands:
        if error["textname"] == name:
            return error['shortdesc'] + '\r\n'
            break

    # Comando no encontrado
    raise Exception("Method with name '%s' not found" % name)


def getRegistro(name):
    found = 0
    for registro in commands:
        if registro["textname"] == name:
            found = 1
            return registro['name'] + '\r\n'
            break
    if not found:
        # Registro no encontrado
        return 'Not found'


def getParams():
    # Devuelve los valores de los registros de control
    global data
    global send_data
```

```
global resp
global new_resp
global new_msg
global getP
error = 0

try:
    for i in range(6):
        data = bytes('ATS0' + str(i) + '?\r\n', 'utf-8')
        send_data = 1

        while not new_msg:
            pass

        if 'ERROR' in str(msg):
            error = 1
            resp = msg
            new_msg = 0
            new_resp = 1
            time.sleep(0.1)
            break
        else:
            new_msg = 0
            s = str(msg).split('\\r\\n')[1]
            name = getRegistro('S0'+str(i))
            name = name.split('\r')[0]
            m = 'S0' + str(i) + ' - ' + name + ': ' + s + '\r\n'
            resp = bytes(m,'utf-8')
            new_resp = 1
            time.sleep(0.1)

    if not error:
        data = bytes('ATS0A?\r\n','utf-8')
        send_data = 1

        while not new_msg:
            pass

        if 'ERROR' in str(msg):
            error = 1
            resp = msg
            new_msg = 0
            new_resp = 1
            time.sleep(0.1)
        else:
            new_msg = 0
            s = str(msg).split('\\r\\n')[1]
            name = getRegistro('S0A')
            name = name.split('\r')[0]
            m = 'S0A - ' + name + ': ' + s + '\r\n'
            resp = bytes(m,'utf-8')
            new_resp = 1
            time.sleep(0.1)
    if not error:
        data = bytes('ATS0B?\r\n','utf-8')
        send_data = 1

        while not new_msg:
            pass

        if 'ERROR' in str(msg):
            error = 1
            resp = msg
            new_msg = 0
            new_resp = 1
            time.sleep(0.1)
        else:
            new_msg = 0
            s = str(msg).split('\\r\\n')[1]
            name = getRegistro('S0B')
            name = name.split('\r')[0]
            m = 'S0B - ' + name + ': ' + s + '\r\n'
            resp = bytes(m,'utf-8')
            new_resp = 1
```

```
                time.sleep(0.1)
            if not error:
                for i in range(10,20):
                    data = bytes('ATS' + str(i) + '?\r\n','utf-8')
                    send_data = 1

                    while not new_msg:
                        pass

                    if 'ERROR' in str(msg):
                        error = 1
                        resp = msg
                        new_msg = 0
                        new_resp = 1
                        time.sleep(0.1)
                        break
                    else:
                        new_msg = 0
                        s = str(msg).split('\\r\\n')[1]
                        name = getRegistro('S'+str(i))
                        4
                        name = name.split('\r')[0]
                        m = 'S' + str(i) + ' - ' + name + ': ' + s + '\r\n'
                        resp = bytes(m,'utf-8')
                        new_resp = 1
                        time.sleep(0.1)
            if not error:
                # Manda mensaje indicando que ha terminado la orden solicitada
                resp = bytes('\r\n\r\nDONE!\r\n', 'utf-8')
                new_resp = 1
                time.sleep(0.1)

            getP = 0

        except:
            m = 'Se ha producido un error. Vuelva a intentarlo. \r\n'
            resp = bytes(m,'utf-8')
            new_resp = 1
            time.sleep(0.1)


    def setParams(*args, **kwargs):
        # Modifica los valores de los registros de control indicados
        global resp
        global new_resp
        global data
        global send_data
        global new_msg
        global setP
        error = 0
        try:
            # Configura el dispositivo al modo 00
            data = getCommandByName('AT+MODE_write')
            data = data.split('<')[0]
            data = bytes(data + '00\r\n','utf-8')
            send_data = 1
            resp = bytes('\r\n--MODE = 00\r\n', 'utf-8')
            new_resp = 1
            time.sleep(0.1)

            while not new_msg:
                pass

            resp = msg
            new_msg = 0
            new_resp = 1
            time.sleep(0.1)

            if not 'ERROR' in str(resp):
                # Cambia los valores de los registros indicados
                params = args[0].split(',')
                for param in params:
                    reg = param.split('=')
                    if getRegistro(reg[0]) == 'Not found':
```

```
                    m = 'ERROR: {!r} no existe\r\n\r\n'.format(reg[0])
                    resp = bytes(m, 'utf-8')
                    new_resp = 1
                    time.sleep(0.1)
                    error = 1
                    break
                else:
                    data = 'AT' + param + '\r\n'
                    data = bytes(data, 'utf-8')
                    send_data = 1
                    m = '\r\n--' + param + '\r\n'
                    resp = bytes(m, 'utf-8')
                    new_resp = 1
                    time.sleep(0.1)

                    while not new_msg:
                        pass

                    resp = msg
                    new_msg = 0
                    new_resp = 1
                    time.sleep(0.1)

                    if 'ERROR' in str(resp):
                        error = 1
                        break

            if not error:
                # Manda mensaje indicando que ha terminado la orden solicitada
                resp = bytes('\r\n\r\nDONE!\r\n', 'utf-8')
                new_resp = 1
                time.sleep(0.1)

        setP = 0

    except:
        m = 'Se ha producido un error. Vuelva a intentarlo. \r\n'
        resp = bytes(m, 'utf-8')
        new_resp = 1
        time.sleep(0.1)


def establishNetwork():
    # Crea la red y recluta nodos
    global data
    global send_data
    global resp
    global new_resp
    global new_msg
    global establish

    try:
        # Configura el dispositivo al modo 01 para poder establecer la red
        data = getCommandByName('AT+MODE_write')
        data = data.split('<')[0]
        data = data + '01\r\n'
        data = bytes(data, 'utf-8')
        send_data = 1
        resp = bytes('\r\n--MODE = 01\r\n', 'utf-8')
        new_resp = 1
        time.sleep(0.1)

        while not new_msg:
            pass

        resp = msg
        new_msg = 0
        new_resp = 1
        time.sleep(0.1)

        if not 'ERROR' in str(resp):
            # Crea la red segun los parametros establecidos en los registros S00, S01, S02. S03
            data = getCommandByName('establish_network')
            data = bytes(data, 'utf-8')
```

```
send_data = 1
resp = bytes('\r\n--ESTABLISH NETWORK:\r\n', 'utf-8')
new_resp = 1
time.sleep(0.1)

while not new_msg:
   pass

resp = msg
new_msg = 0
new_resp = 1
time.sleep(0.1)

if not 'ERROR' in str(resp):
   # Recluta nodos
   data = getCommandByName('recruiting_process')
   data = bytes(data, 'utf-8')
   send_data = 1
   resp = bytes('\r\n--RECRUITING PROCESS:\r\n', 'utf-8')
   new_resp = 1
   time.sleep(0.1)

   while not new_msg:
      pass

   if not 'ERROR' in str(msg):
      #La respuesta de reclutar nodos termina con ready.
      while not 'READY' in str(msg):
         if new_msg and not 'READY' in str(msg):
            resp = msg
            new_resp = 1
            new_msg = 0
            time.sleep(0.1)
      # Manda el ready
      resp = msg
      new_msg = 0
      new_resp = 1
      time.sleep(0.1)

      # Muestra la tabla con los nodos que estan en la red y sus direcciones
      data = getCommandByName('network_table')
      data = bytes(data, 'utf-8')
      send_data = 1
      resp = bytes('\r\n--NETWORK TABLE:\r\n', 'utf-8')
      new_resp = 1
      time.sleep(0.1)

      while not new_msg:
         pass

      if not 'ERROR' in str(msg):
         while not 'OK' in str(msg):
            if new_msg and not 'OK' in str(msg):
               resp = msg
               new_resp = 1
               new_msg = 0
               time.sleep(0.1)
         # Manda el ok
         resp = msg
         new_msg = 0
         new_resp = 1
         time.sleep(0.1)

         # Manda mensaje indicando que ha terminado la orden solicitada
         resp = bytes('\r\n\r\nDONE!\r\n', 'utf-8')
         new_resp = 1
         time.sleep(0.1)
      else:
         # Mensaje de error
         resp = msg
         new_msg = 0
         new_resp = 1
         time.sleep(0.1)
   else:
```

```
                # Mensaje de error
                resp = msg
                new_msg = 0
                new_resp = 1
                time.sleep(0.1)
            establish = 0

        except:
            m = 'Se ha producido un error. Vuelva a intentarlo. \r\n'
            resp = bytes(m,'utf-8')
            new_resp = 1
            time.sleep(0.1)


    def destroyNetwork():
        # Destruye la red
        global data
        global send_data
        global resp
        global new_resp
        global new_msg
        global destroy

        try:
            # Configura el dispositivo al modo 01
            data = getCommandByName('AT+MODE_write')
            data = data.split('<')[0]
            data = data + '01\r\n'
            data = bytes(data, 'utf-8')
            send_data = 1
            resp = bytes('\r\n--MODE = 01\r\n', 'utf-8')
            new_resp = 1
            time.sleep(0.1)

            while not new_msg:
                pass

            resp = msg
            new_msg = 0
            new_resp = 1
            time.sleep(0.1)

            if not 'ERROR' in str(resp):
                # Destruye la red existente
                data = getCommandByName('destroy_network')
                data = bytes(data, 'utf-8')
                send_data = 1
                resp = bytes('\r\n--DESTROY NETWORK\r\n', 'utf-8')
                new_resp = 1
                time.sleep(0.1)

                while not new_msg:
                    pass

                resp = msg
                new_msg = 0
                new_resp = 1
                time.sleep(0.1)

                if not 'ERROR' in str(resp):
                    resp = bytes('\r\n\r\nDONE!\r\n', 'utf-8')
                    new_resp = 1
                    time.sleep(0.1)

            destroy = 0

        except:
            m = 'Se ha producido un error. Vuelva a intentarlo. \r\n'
            resp = bytes(m,'utf-8')
            new_resp = 1
            time.sleep(0.1)
```

```
def updateNetwork():
    # Comprueba el estado de la red, actualiza la tabla de red y recluta nuevos nodos
    global data
    global send_data
    global resp
    global new_resp
    global new_msg
    global update

    try:
        # Configura el dispositivo al modo 01
        data = getCommandByName('AT+MODE_write')
        data = data.split('<')[0]
        data = data + '01\r\n'
        data = bytes(data, 'utf-8')
        send_data = 1
        resp = bytes('\r\n--MODE = 01\r\n', 'utf-8')
        new_resp = 1
        time.sleep(0.1)

        while not new_msg:
            pass

        resp = msg
        new_msg = 0
        new_resp = 1
        time.sleep(0.1)

        if not 'ERROR' in str(resp):
            # Devuelve el estado de la conexion de la red
            data = getCommandByName('network_state')
            data = bytes(data, 'utf-8')
            send_data = 1
            resp = bytes('\r\n--NETWORK STATE:\r\n', 'utf-8')
            new_resp = 1
            time.sleep(0.1)

            while not new_msg:
                pass

            resp = msg
            new_msg = 0
            new_resp = 1
            time.sleep(0.1)

            if not 'ERROR' in str(resp):
                if 'NoNET' in str(resp):
                    # Manda mensaje indicando que ha terminado la orden solicitada
                    resp = bytes('\r\n\r\nDONE!\r\n', 'utf-8')
                    new_resp = 1
                    time.sleep(0.1)
                else:
                    # Actualiza la tabla de la red. Supone master enhanced
                    data = getCommandByName('update_network_table')
                    data = data.split('[')[0] + '\r\n'
                    data = bytes(data, 'utf-8')
                    send_data = 1
                    resp = bytes('\r\n--UPDATE NETWORK TABLE:\r\n', 'utf-8')
                    new_resp = 1
                    time.sleep(0.1)

                    while not new_msg:
                        pass

                    if not 'ERROR' in str(msg):
                        while not 'READY' in str(msg):
                            if new_msg and not 'READY' in str(msg):
                                resp = msg
                                new_resp = 1
                                new_msg = 0
                                time.sleep(0.1)
                        # Manda el ready
                        resp = msg
                        new_msg = 0
```

```
            new_resp = 1
            time.sleep(0.05)

            # Recluta nodos
            resp = bytes('\r\n--RECRUITING PROCESS:\r\n', 'utf-8')
            new_resp = 1
            data = getCommandByName('recruiting_process')
            data = bytes(data, 'utf-8')
            send_data = 1
            time.sleep(0.1)

            while not new_msg:
                pass

            if not 'ERROR' in str(msg):
                #La respuesta de reclutar nodos termina con ready.
                while not 'READY' in str(msg):
                    if new_msg and not 'READY' in str(msg):
                        resp = msg
                        new_resp = 1
                        new_msg = 0
                        time.sleep(0.1)
                # Manda el ready
                resp = msg
                new_msg = 0
                new_resp = 1
                time.sleep(0.1)

                # Muestra la tabla con los nodos que estan en la red y sus direccion
                resp = bytes('\r\n--NETWORK TABLE:\r\n', 'utf-8')
                new_resp = 1
                data = getCommandByName('network_table')
                data = bytes(data, 'utf-8')
                send_data = 1
                time.sleep(0.1)

                while not new_msg:
                    pass

                if not 'ERROR' in str(msg):
                    while not 'OK' in str(msg):
                        if new_msg and not 'OK' in str(msg):
                            resp = msg
                            new_resp = 1
                            new_msg = 0
                            time.sleep(0.1)
                    # Manda el ok
                    resp = msg
                    new_msg = 0
                    new_resp = 1
                    time.sleep(0.1)

                    # Manda mensaje indicando que ha terminado la orden solicitada
                    resp = bytes('\r\n\r\nDONE!\r\n', 'utf-8')
                    new_resp = 1
                    time.sleep(0.1)
                else:
                    # Mensaje de error
                    resp = msg
                    new_msg = 0
                    new_resp = 1
                    time.sleep(0.1)
            else:
                # Mensaje de error
                resp = msg
                new_msg = 0
                new_resp = 1
                time.sleep(0.1)
        else:
            # Mensaje de error
            resp = msg
            new_msg = 0
            new_resp = 1
            time.sleep(0.1)
```

```
            update = 0

        except:
            m = 'Se ha producido un error. Vuelva a intentarlo. \r\n'
            resp = bytes(m,'utf-8')
            new_resp = 1
            time.sleep(0.1)


    def sendMessage(addr, XX, message):
        # Envia el mensaje unicast al nodo destino
        global data
        global send_data
        global resp
        global new_resp
        global new_msg
        global send

        try:
            # Configura el dispositivo al modo 01
            data = getCommandByName('AT+MODE_write')
            data = data.split('<')[0]
            data = data + '01\r\n'
            data = bytes(data, 'utf-8')
            send_data = 1
            resp = bytes('\r\n--MODE = 01\r\n', 'utf-8')
            new_resp = 1
            time.sleep(0.1)

            while not new_msg:
                pass

            resp = msg
            new_msg = 0
            new_resp = 1
            time.sleep(0.1)

            if not 'ERROR' in str(resp):
                # Actualiza la tabla de red
                data = getCommandByName('update_network_table')
                data = data.split('[')[0] + '\r\n'
                data = bytes(data, 'utf-8')
                send_data = 1
                resp = bytes('\r\n--ADDRESS IN NETWORK TABLE?\r\n', 'utf-8')
                new_resp = 1
                time.sleep(0.1)

                while not new_msg:
                    pass

                if not 'ERROR' in str(msg):
                    nodes = []
                    while not 'READY' in str(msg):
                        if new_msg and not 'READY' in str(msg):
                            resp = msg
                            new_msg = 0
                            time.sleep(0.1)
                            if 'NTE' in str(resp):
                                nodes.append(str(resp).split(':')[1])
                    # Recibe el ready
                    resp = msg
                    new_msg = 0
                    time.sleep(0.1)

                    #Comprueba si la direccion indicada pertenece a un nodo de la red
                    found = 0
                    for node in nodes:
                        if addr in node:
                            resp = bytes('\r\nOK\r\n', 'utf-8')
                            new_resp = 1
                            time.sleep(0.1)

                            # Manda mensaje
```

```
                        resp = bytes('\r\n--SEND UNICAST:\r\n', 'utf-8')
                        new_resp = 1
                        time.sleep(0.1)

                        dig = int(len(message)/ int(XX))
                        data = getCommandByName('msg_to_node')
                        data = data.split('<')[0] + str(addr) + ',' + XX + ','
                        for i in range(int(XX)):
                            j = dig*(i+1)
                            d = '0x' + message[i:j]
                            data = data + d

                        data = data + '\r\n'
                        data = bytes(data, 'utf-8')
                        send_data = 1

                        while not new_msg:
                            pass

                        if not 'ERROR' in str(msg):
                            while not 'ACK' in str(msg):
                                if new_msg and not 'ACK' in str(msg):
                                    resp = msg
                                    new_resp = 1
                                    new_msg = 0
                                    time.sleep(0.1)
                            resp = msg
                            new_msg = 0
                            new_resp = 1
                            time.sleep(0.1)

                            # Manda mensaje indicando que ha terminado la orden solicitada
                            resp = bytes('\r\n\r\nDONE!\r\n', 'utf-8')
                            new_resp = 1
                            time.sleep(0.1)

                        else:
                            # Mensaje de error
                            resp = msg
                            new_msg = 0
                            new_resp = 1
                            time.sleep(0.1)
                        found = 1
                        break

                if not found:
                    resp = bytes('\r\nERROR: 10 - No se puede aplicar el comando porque la direccion
                                  indicada no corresponde a un nodo de la red\r\n', 'utf-8')
                    new_resp = 1
                    time.sleep(0.1)
            send = 0
        except Exception as e:
            print(e)
            m = 'Se ha producido un error. Vuelva a intentarlo. \r\n'
            resp = bytes(m,'utf-8')
            new_resp = 1
            time.sleep(0.1)


    def sendBroadcast(message):
        # Envia el mensaje broadcast a todos los nodos de la red
        global data
        global send_data
        global resp
        global new_resp
        global new_msg
        global broadcast

        try:
            # Configura el dispositivo al modo 01
            data = getCommandByName('AT+MODE_write')
            data = data.split('<')[0]
            data = data + '01\r\n'
            data = bytes(data, 'utf-8')
```

```
            send_data = 1
            resp = bytes('\r\n--MODE = 01\r\n', 'utf-8')
            new_resp = 1
            time.sleep(0.1)

            while not new_msg:
                pass

            resp = msg
            new_msg = 0
            new_resp = 1
            time.sleep(0.1)

            if not 'ERROR' in str(resp):
                #Envia el mensaje a todos los nodos
                resp = bytes('\r\n--SEND BROADCAST:\r\n', 'utf-8')
                new_resp = 1
                time.sleep(0.1)

                if len(message) < 10:
                    XX = str(0) + str(len(message))
                else:
                    XX = str(len(message))

                data = getCommandByName('broadcast_msg')
                data = data.split(':')[0] + ':' + str(XX) + ',' + message + '\r\n'
                data = bytes(data, 'utf-8')
                send_data = 1

                while not new_msg:
                    pass

                resp = msg
                new_msg = 0
                new_resp = 1
                time.sleep(0.1)

                if not 'ERROR' in str(resp):
                    # Manda mensaje indicando que ha terminado la orden solicitada
                    resp = bytes('\r\n\r\nDONE!\r\n', 'utf-8')
                    new_resp = 1
                    time.sleep(0.1)

            broadcast = 0
        except Exception as e:
            m = 'Se ha producido un error. Vuelva a intentarlo. \r\n'
            resp = bytes(m,'utf-8')
            new_resp = 1
            time.sleep(0.1)


    def sendBeacon(message):
        # Envia el beacon a todos los nodos que puedan escucharle
        global data
        global send_data
        global resp
        global new_resp
        global new_msg
        global beacon

        try:
            # Configura el dispositivo al modo 01
            data = getCommandByName('AT+MODE_write')
            data = data.split('<')[0]
            data = data + '01\r\n'
            data = bytes(data, 'utf-8')
            send_data = 1
            resp = bytes('\r\n--MODE = 01\r\n', 'utf-8')
            new_resp = 1
            time.sleep(0.1)

            while not new_msg:
                pass
```

```
            resp = msg
            new_msg = 0
            new_resp = 1
            time.sleep(0.1)

            if not 'ERROR' in str(resp):
                # Manda mensaje
                resp = bytes('\r\n--SEND BEACON:\r\n', 'utf-8')
                new_resp = 1
                time.sleep(0.1)

                XX = str(hex(len(message)))
                XX = XX.split('x')[1]

                if len(XX) < 2:
                    XX = str(0) + XX

                data = getCommandByName('beacon_msg')
                data = data.split(':')[0] + ':' + XX + ',' + message + '\r\n'
                data = bytes(data, 'utf-8')
                send_data = 1

                while not new_msg:
                    pass

                resp = msg
                new_msg = 0
                new_resp = 1
                time.sleep(0.1)

                if not 'ERROR' in str(resp):
                    # Manda mensaje indicando que ha terminado la orden solicitada
                    resp = bytes('\r\n\r\nDONE!\r\n', 'utf-8')
                    new_resp = 1
                    time.sleep(0.1)
            beacon = 0
        except Exception as e:
            m = 'Se ha producido un error. Vuelva a intentarlo. \r\n'
            resp = bytes(m,'utf-8')
            new_resp = 1
            time.sleep(0.1)


    def receivedBeacon():
        global resp
        global new_resp
        global new_beacon
        global json_tag

        num_data = 16 #96 bytes -> 48 datos -> 16 datos por eje
        message_id = 0
        latestBeaconTimestamp_nano = None
        sampleTime = 0
        stepTime = 10*10**6 # El sensor manda mensajes cada 10 milisegundos -> nanosegundos
        macAddress = None
        msg_data = None
        id_comp = 0

        while 1:
            if stop_threads:
                break
            try:
                if new_beacon and '=' in str(msg):
                    msgs = msg.split(b'BEACONB:')

                    # Timestamp
                    t = float("%.9f" % time.time())
                    t_nano = int(t * 10**9) # nanosegundos

                    for i in range(len(msgs)):
                        samples=[]
                        if i == 0:
                            continue
```

```
# MAC del sensor
macAddress = msgs[i][0:12]
# Datos recibidos
m = msgs[i].split(b'\r\n')[0]
end_data = len(m) - 10
m = msgs[i][16:end_data]

if msgs[i][13:15] == b'61':
    # Beacon con identificador
    message_id = m[0]

    while not message_id == id_comp:
        # Se ha perdido algun paquete -> rellena con ceros ese paquete
        if id_comp == 0:
            latestBeaconTimestamp_nano = t_nano

        sampleTime = latestBeaconTimestamp_nano + id_comp * stepTime * 16

        # Guarda los datos en una lista de muestras
        for i in range(num_data):
            x = 0
            y = 0
            z = 0
            samples.append({'id':id_comp,'x':x,'y':y,'z':z,'sampleTime':sampeTime})
            sampleTime = sampleTime + stepTime

        if id_comp == 14:
            id_comp = 0
        else:
            id_comp = id_comp + 1

    msg_data = m[1:]

    if message_id == 0:
        latestBeaconTimestamp_nano = t_nano
    sampleTime = latestBeaconTimestamp_nano + message_id * stepTime * 16

# Decodifica los datos(96 bytes -> 2 por dato)
message = struct.unpack('<hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh', msg_data)

# Guarda los datos en una lista de muestras
for i in range(num_data):
    j = i*3
    x = message[j]
    y = message[j+1]
    z = message[j+2]
    samples.append({'id':id_comp,'x':x,'y':y,'z':z,'sampleTime':sampleTime})
    sampleTime = sampleTime + stepTime

# Guarda los datos y los sube a influxDB
json_body = []

for sample in samples:
    id_msg = sample.get("id")
    x = sample.get("x")
    y = sample.get("y")
    z = sample.get("z")
    t_sample = sample.get("sampleTime")

    json_body = [
        {
            "measurement": "raw_mag",
            "tags": {
                "mac": str(macAddress),
                "beacon": str(latestBeaconTimestamp_nano)
            },
            "time": t_sample,
            "fields": {
                "x": x,
                "y": y,
                "z": z,
                "packet_id": id_msg,
                "received_at": t_nano
            }
```

```
                        }
                    ]

                    json_influx.append(json_body) # lista de datos que se suben a influxdb

                if id_comp == 14:
                    id_comp = 0
                else:
                    id_comp = id_comp + 1

            new_beacon = 0
            time.sleep(0.005)

        except Exception as e:
            print('receivedBeacon:', e)
            m = 'Error al recibir beacon. \r\n'
            resp = bytes(m,'utf-8')
            new_resp = 1
            time.sleep(0.01)


def iniInflux(host,port,dbname):
    global client_influx
    try:
        #Inicializacion de objeto Influx
        client_influx = InfluxDBClient(host, port)
        # Comprueba si existe la base de datos. Si no existe la crea
        list_db = client_influx.get_list_database()
        found = 0

        for db in list_db:
            if dbname in db['name']:
                print('Existe la base de datos ', dbname_influx)
                found = 1
                break

        if found == 0:
            print("Base de datos creada: " + dbname_influx)
            client_influx.create_database(dbname_influx)

        #Cambiar a base de datos
        print("Base de datos: " + dbname)
        client_influx.switch_database(dbname)

    except Exception as e:
        print('InfluxDB:', e)
        return 1

    return 0


def influx_upload():
    global json_influx
    global resp
    global new_resp

    while 1:
        if stop_threads:
            break

        try:
            if len(json_influx) >= 48:
                client_influx.write_points(json_influx[0:48])
                for i in range(48):
                    json_influx.pop(0)

        except Exception as e:
            print('influx_upload:', e)
            m = 'Error al subir los datos a influxdb. \r\n'
            resp = bytes(m,'utf-8')
            new_resp = 1
            time.sleep(0.01)
```

```
# MAIN:
host_influx = '155.210.139.83'
port_influx = 8086
dbname_influx = 'sensor_mag'
client_influx = ''

msg = ''
data = bytes('','utf-8')
resp = ''
new_data = 0
send_data = 1
new_msg = 0
new_resp = 0
getP = 0
setP = 0
establish = 0
destroy = 0
update = 0
send = 0
beacon = 0
broadcast = 0
new_beacon = 0
connection = None
stop_threads = 0
json_tag = []
json_influx = []

# Lee el fichero de comandos "loraapi.json"
with open('loraapi.json', encoding='utf-8') as f:
    lora = json.load(f, strict=False)

groups = lora['groups']
# Almacena todos los comandos en una lista
commands = []
for group in groups:
    for c in group['commands']:
        commands.append(c)

# Accede al puerto serie (LoRa)
p_serie = serial.Serial('/dev/ttyUSB0', 115200)

# Crea el socket TCP/IP
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = ('localhost',10000)
server_address = ('192.168.0.115',10000)
sock.bind(server_address)
#Escucha nuevas conexiones entrantes
sock.listen(1)

# Crea los threads que se encargan de recibir y enviar por el socket y por el puerto serie,
# de recibir los beacon del sensor y de enviar los datos al cliente InfluxDB
thread_recv_sock = threading.Thread(target=socket_server_recv)
thread_send_sock = threading.Thread(target=socket_server_send)
thread_serial = threading.Thread(target=com_serial)
thread_beacon = threading.Thread(target=receivedBeacon)
thread_influx = threading.Thread(target=influx_upload)

try:
    thread_recv_sock.start()
    thread_send_sock.start()
    thread_serial.start()
    thread_beacon.start()

    # inicializa el nodo a modo 01
    data = getCommandByName('AT+MODE_write')
    data = data.split('<')[0]
    data = data + '01\r\n'
    data = bytes(data, 'utf-8')
    send_data = 1
    time.sleep(0.1)

    # Inicializa influx
    init_influx = iniInflux(host_influx, port_influx, dbname_influx)
```

```
        if init_influx == 1:
            print("Se ha producido un error al iniciar influx")
        else:
            thread_influx.start()

        print('Gestor preparado')

        while 1:
            if new_data:
                new_data = 0
                if 'getParams()' in str(data):
                    getP = 1
                    getParams()
                elif 'setParams(' in str(data):
                    setP = 1
                    data = str(data).split('(')[1]
                    data = data.split(')')[0]
                    setParams(data)
                elif 'establishNetwork()' in str(data):
                    establish = 1
                    establishNetwork()
                elif 'destroyNetwork()' in str(data):
                    destroy = 1
                    destroyNetwork()
                elif 'updateNetwork' in str(data):
                    update = 1
                    updateNetwork()
                elif 'sendMessage(' in str(data):
                    send = 1
                    p = str(data).split('(')[1]
                    p = p.split(')')[0]
                    p = p.split(',')
                    sendMessage(p[0], p[1], p[2])
                elif 'sendBroadcast(' in str(data):
                    broadcast = 1
                    p = str(data).split('(')[1]
                    p = p.split(')')[0]
                    sendBroadcast(p)
                elif 'sendBeacon(' in str(data):
                    beacon = 1
                    p = str(data).split('(')[1]
                    p = p.split(')')[0]
                    sendBeacon(p)
                else:
                    data = str(data).split("\\r")[0]
                    data = str(data).split("'")[1] + '\r\n'
                    if not data == '\r\n':
                        data = bytes(data, 'utf-8')
                        send_data = 1

    except KeyboardInterrupt:
        print('Terminando procesos')
        sock.close()
        stop_threads = 1
        thread_recv_sock.join()
        thread_send_sock.join()
        thread_serial.join()
        thread_beacon.join()

        print('Gestor parado')

    # Cierra el puerto serie
    p_serie.close()
```

Anexo F. Código para actualizar la base de datos

```
import pandas as pd
import numpy as np
import time
import datetime
import os
from ftplib import FTP

# Conexion con el servidor FTP
ftp = FTP()
ftp.set_debuglevel(1)
ftp.connect('howlab.i3a.es', 21)
ftp.login('howlab', 'ppito56')
ftp.cwd('/base_datos')


def ftp_upload(localfile, remotefile):
    fp = open(localfile, 'rb')
    try:
        ftp.storbinary('STOR %s' % remotefile, fp, 1024)
    except Exception:
        # No existe el directorio -> lo crea
        path, filename = os.path.split(remotefile)
        ftp.mkd(path)
        ftp_upload(localfile,remotefile)
        fp.close()
        return
    fp.close()


data = pd.read_csv('G:/Mi unidad/TFM/aprendizaje/datos_influx.csv',delimiter=',')

# Resta el ruido de fondo en cada eje
x = data['raw_mag.x'].values - np.mean(data['raw_mag.x'].values)
y = data['raw_mag.y'].values - np.mean(data['raw_mag.y'].values)
z = data['raw_mag.z'].values - np.mean(data['raw_mag.z'].values)

samples_t = data['time'].values
samples_time = []
for i in range(len(samples_t)):
    aux = str(samples_t[i])
    aux = aux.replace('T', ' ')
    aux = aux.replace('Z', '')
    samples_time.append(aux)

x_vehicle = []
y_vehicle = []
z_vehicle = []
timestamp_vehicle = []
date_vehicle = []
vehicle = 0
cont = 0
vehicles = []

for i in range(20, len(x)-20):
    # Si se supera el umbral, se guardan desde las 20 muestras anteriores hasta
    # las 20 muestras sin que se supere el umbral

    if not vehicle:
        if np.abs(x[i]) > 50 and np.abs(y[i]) > 50 and np.abs(z[i]) > 50:
            for j in range(20,0,-1):
                x_vehicle.append(x[i-j])
                y_vehicle.append(y[i-j])
                z_vehicle.append(z[i-j])
                date_vehicle.append(samples_time[i-j])
                t = time.mktime(datetime.datetime.strptime(samples_time[i-j],
                                                           "%Y-%m-%d %H:%M:%S.%f").timetuple())
                timestamp_vehicle.append(t)
```

```
            x_vehicle.append(x[i])
            y_vehicle.append(y[i])
            z_vehicle.append(z[i])
            date_vehicle.append(samples_time[i])
            t = time.mktime(datetime.datetime.strptime(samples_time[i], "%Y-%m-%d %H:%M:%S.%f").
            timestamp_vehicle.append(t)
            vehicle = 1

        else:
            x_vehicle.append(x[i])
            y_vehicle.append(y[i])
            z_vehicle.append(z[i])
            date_vehicle.append(samples_time[i])
            t = time.mktime(datetime.datetime.strptime(samples_time[i], "%Y-%m-%d %H:%M:%S.%f").timetuple())
            timestamp_vehicle.append(t)

            if np.abs(x[i]) < 50 and np.abs(y[i]) < 50 and np.abs(z[i]) < 50:
                # Si llega el contador a 20 -> no hay vehiculo
                if cont == 19:
                    vehicle = 0
                    cont = 0
                    data_sensor = {'raw_mag.x': x_vehicle.copy(), 'raw_mag.y': y_vehicle.copy(),
                                   'raw_mag.z': z_vehicle.copy(), 'time': date_vehicle.copy(),
                                   'timestamp': timestamp_vehicle.copy()}
                    if len(x) > 60:
                        # Para disminuir las falsas alarmas: al menos 20 muestras deben superar el u
                        vehicles.append(data_sensor)

                    x_vehicle.clear()
                    y_vehicle.clear()
                    z_vehicle.clear()
                    date_vehicle.clear()
                    timestamp_vehicle.clear()

                cont += 1
            else:
                cont = 0


    for i in range(len(vehicles)):
        # Obtiene lista con todas las etiquetas sin asignar
        labels = ftp.nlst('/labels')

        for j in range(len(labels)):
            # Busca una etiqueta con un timestamp igual al registrado para la huella
            #margen de diferencia entre timestamp: 1 segundo
            timestamp_sensor = int(vehicles[i]['timestamp'][0])
            label = labels[j].split('.')[0] # Quita '.json'
            timestamp_label = int(label)
            diff = np.abs(timestamp_sensor - timestamp_label)

            if diff < 2:
                # Crea el fichero con los datos del sensor y con el nombre igual que el de la etiqueta
                # y lo manda al servidor. Mueve la etiqueta y el video a las carpetas de la base de datos

                data_veh = pd.DataFrame(vehicles[i]).drop('timestamp',1)
                data_veh.to_csv(label + '.csv')

                # Sube el fichero al servidor y lo borra de la maquina local
                path_data = '/base_datos/data/' + label + '.csv'
                ftp_upload(label + '.csv', path_data)
                os.remove(label + '.csv')

                # Mueve el video y la etiqueta a la base de datos
                ftp.rename('/labels/' + labels[j], '/base_datos/labels/' + labels[j])
                ftp.rename('/videos/' + label + '.mp4', '/base_datos/videos/' + label + '.mp4')

                #Etiqueta encontrada
                break

    # Cierra conexion con servidor ftp
    ftp.quit()
```

Anexo G. Código para el tratamiento de los datos

```
import numpy as np
import json
import os
import pandas as pd
from scipy import signal as signal
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# Inicializa variables
len_vehicles = []
speed_vehicles = []
direction_vehicles = []
data_sensor = []
names = []
mean_x = []
mean_y = []
mean_z = []
std_x = []
std_y = []
std_z = []
num_samples = 20

path = 'G:/Mi unidad/TFM/aprendizaje/BASE_DATOS/'
files = os.listdir(path)

# Lee los datos
idx = [(indice)for indice, string in enumerate(files) if "data" in string][0]
f = os.listdir(path + files[idx])

for j in range(len(f)):
    local_path = path + files[idx] + '/' + f[j]

    # Guarda los nombres para leer despues las etiquetas en el mismo orden
    names.append(f[j].split('.')[0])

    # Extrae los datos de cada eje.
    # Hace un remuestreo para que las ventanas temporales tengan un tamaño fijo
    # Concatena las tres señales para hacer una unica ventana, junto con la media y la
    # desviacion estandar de cada eje

    data_aux = pd.read_csv(local_path,delimiter=';')

    # EJE X
    data_aux_x = data_aux['raw_mag.x'].values
    # Calcula la desviacion estandar de la señal
    s_x = np.std(data_aux_x)
    std_x.append(s_x)
    # Calcula la media de la señal y la resta
    m_x = np.mean(data_aux_x, dtype=np.int64)
    mean_x.append(m_x)
    # Normalizacion gaussiana: restar la media y dividir por la std
    data_aux_x -= m_x
    data_x = data_aux_x / s_x
    # Remuestrea la señal
    data_x = signal.resample(data_x, num_samples)

    # EJE Y
    data_aux_y = data_aux['raw_mag.y'].values
    # Calcula la desviacion estandar de la señal
    s_y = np.std(data_aux_y)
    std_y.append(s_y)
    # Calcula la media de la señal y la resta
    m_y = np.mean(data_aux_y, dtype=np.int64)
    mean_y.append(m_y)
    # Normalizacion gaussiana: restar la media y dividir por la std
    data_aux_y -= m_y
    data_y = data_aux_y / s_y
    # Remuestrea la señal
```

```
    data_y = signal.resample(data_y, num_samples)

    # EJE Z
    data_aux_z = data_aux['raw_mag.z'].values
    # Calcula la desviacion estandar de la señal
    s_z = np.std(data_aux_z)
    std_z.append(s_z)
    # Calcula la media de la señal y la resta
    m_z = np.mean(data_aux_z, dtype=np.int64)
    mean_z.append(m_z)
    # Normalizacion gaussiana: restar la media y dividir por la std
    data_aux_z -= m_z
    data_z = data_aux_z / s_z
    # Remuestrea la señal
    data_z = signal.resample(data_z, num_samples)

    # Concatena las señales de los tres ejes
    data_aux_xyz = np.concatenate((data_x, data_y, data_z, np.reshape(m_x, (1,)),
                                   np.reshape(s_x, (1,)), np.reshape(m_y,(1,)),
                                   np.reshape(s_y,(1,)), np.reshape(m_z,(1,)),
                                   np.reshape(s_z,(1,))))
    data_aux_xyz = np.reshape(data_aux_xyz, (len(data_aux_xyz),1))
    data_sensor.append(data_aux_xyz)

data_sensor = np.asarray(data_sensor)
data_sensor = np.reshape(data_sensor, (data_sensor.shape[0], data_sensor.shape[1]))

# Guarda un fichero con los datos
data_s = pd.DataFrame(data_sensor)
data_s.to_csv('G:/Mi unidad/TFM/aprendizaje/data_xyz_' + str(num_samples) + '_mean_std.csv')


# Lee las etiquetas en el orden en el que ha leido los datos
idx = [(indice)for indice, string in enumerate(files) if "labels" in string][0]
f = os.listdir(path + files[idx])

for i in range(len(names)):
    for j in range(len(f)):
        if names[i] == f[j].split('.')[0]:
            local_path = path + files[idx] + '/' + f[j]

            with open(local_path) as vehicle_data:
                data = json.load(vehicle_data)
            len_vehicles.append(data['vehiculo'][0]['Tamagno (metros)'])
            speed_vehicles.append(data['vehiculo'][0]['Velocidad (km/h)'])
            direction_vehicles.append(data['vehiculo'][0]['Direccion'])
            break

# Modifica las etiquetas:
# direccion cambia a binario: "0" sale, "1" entra
# tamaño y sentido las normaliza entre 0 y 1, de manera que el valor medio sea 0.5

len_min = np.min(len_vehicles)
len_max = np.max(len_vehicles)
speed_min = np.min(speed_vehicles)
speed_max = np.max(speed_vehicles)

y = np.zeros((len(data_sensor),3))

for i in range(len(data_sensor)):
    if 'sale' in direction_vehicles[i]:
        direction_vehicles[i] = 0
    else:
        direction_vehicles[i] = 1

    y[i][0] = 0.5 + (len_vehicles[i] - len_min)/ (len_max - len_min)
    y[i][1] = 0.5 + (speed_vehicles[i] - speed_min) / (speed_max - speed_min)
    y[i][2] = direction_vehicles[i]

df = pd.DataFrame({'Tamagno': y[:,0],
                   'Velocidad': y[:,1],
                   'direccion': y[:,2]})

# Guarda un fichero con las etiquetas
```

```
df.to_csv('G:/Mi unidad/TFM/aprendizaje/labels.csv', index=False)

#%% Correlacion de las medias de los ejes y las etiquetas
mean_x = np.asarray(mean_x)
mean_x = np.reshape(mean_x, (1, len(mean_x)))
mean_y = np.asarray(mean_y)
mean_y = np.reshape(mean_y, (1, len(mean_y)))
mean_z = np.asarray(mean_z)
mean_z = np.reshape(mean_z, (1, len(mean_z)))

len_vehicles = np.asarray(len_vehicles)
len_vehicles = np.reshape(len_vehicles, (1, len(len_vehicles)))
speed_vehicles = np.asarray(speed_vehicles)
speed_vehicles = np.reshape(speed_vehicles, (1, len(speed_vehicles)))
direction_vehicles = np.asarray(direction_vehicles)
direction_vehicles = np.reshape(direction_vehicles, (1, len(direction_vehicles)))

mean_data = np.concatenate((mean_x, mean_y, mean_z, len_vehicles, speed_vehicles,direction_vehicles),0)
mean_data = np.transpose(mean_data)

mean_data_mod = pd.DataFrame(mean_data, columns=['mean_x', 'mean_y', 'mean_z', 'len_vehicles',
                                                 'speed_vehicles', 'direction_vehicles'])

sns.pairplot(mean_data_mod)

# Coeficientes de correlacion
mean_corr_matrix = mean_data_mod.corr()


#%% Correlacion de las desviaciones estandar de los ejes y las etiquetas
std_x = np.asarray(std_x)
std_x = np.reshape(std_x, (1, len(std_x)))
std_y = np.asarray(std_y)
std_y = np.reshape(std_y, (1, len(std_y)))
std_z = np.asarray(std_z)
std_z = np.reshape(std_z, (1, len(std_z)))

std_data = np.concatenate((std_x, std_y, std_z, len_vehicles, speed_vehicles, direction_vehicles), 0)
std_data = np.transpose(std_data)

std_data_mod = pd.DataFrame(std_data, columns=['std_x', 'std_y', 'std_z', 'len_vehicles',
                                               'speed_vehicles', 'direction_vehicles'])

sns.pairplot(std_data_mod)

# Coeficientes de correlacion
std_corr_matrix = std_data_mod.corr()


#%% Separa los datos segun la direccion
mean_x_entra = []
mean_x_sale = []
mean_y_entra = []
mean_y_sale = []
mean_z_entra = []
mean_z_sale = []
std_x_entra = []
std_x_sale = []
std_y_entra = []
std_y_sale = []
std_z_entra = []
std_z_sale = []
direction_entra = []
direction_sale = []
speed_entra = []
speed_sale = []
len_entra = []
len_sale = []

for i in range(direction_vehicles.shape[1]):
    if direction_vehicles[0,i]:
        # vehiculos que entran
        mean_x_entra.append(mean_x[0,i])
        mean_y_entra.append(mean_y[0,i])
```

```
            mean_z_entra.append(mean_z[0,i])

            std_x_entra.append(std_x[0,i])
            std_y_entra.append(std_y[0,i])
            std_z_entra.append(std_z[0,i])

            direction_entra.append(direction_vehicles[0,i])
            speed_entra.append(speed_vehicles[0,i])
            len_entra.append(len_vehicles[0,i])
        else:
            # vehiculos que salen
            mean_x_sale.append(mean_x[0,i])
            mean_y_sale.append(mean_y[0,i])
            mean_z_sale.append(mean_z[0,i])

            std_x_sale.append(std_x[0,i])
            std_y_sale.append(std_y[0,i])
            std_z_sale.append(std_z[0,i])

            direction_sale.append(direction_vehicles[0,i])
            speed_sale.append(speed_vehicles[0,i])
            len_sale.append(len_vehicles[0,i])

    mean_x_entra = np.asarray(mean_x_entra)
    mean_x_entra = np.reshape(mean_x_entra, (1, len(mean_x_entra)))
    mean_y_entra = np.asarray(mean_y_entra)
    mean_y_entra = np.reshape(mean_y_entra, (1, len(mean_y_entra)))
    mean_z_entra = np.asarray(mean_z_entra)
    mean_z_entra = np.reshape(mean_z_entra, (1, len(mean_z_entra)))

    mean_x_sale = np.asarray(mean_x_sale)
    mean_x_sale = np.reshape(mean_x_sale, (1, len(mean_x_sale)))
    mean_y_sale = np.asarray(mean_y_sale)
    mean_y_sale = np.reshape(mean_y_sale, (1, len(mean_y_sale)))
    mean_z_sale = np.asarray(mean_z_sale)
    mean_z_sale = np.reshape(mean_z_sale, (1, len(mean_z_sale)))

    std_x_entra = np.asarray(std_x_entra)
    std_x_entra = np.reshape(std_x_entra, (1, len(std_x_entra)))
    std_y_entra = np.asarray(std_y_entra)
    std_y_entra = np.reshape(std_y_entra, (1, len(std_y_entra)))
    std_z_entra = np.asarray(std_z_entra)
    std_z_entra = np.reshape(std_z_entra, (1, len(std_z_entra)))

    std_x_sale = np.asarray(std_x_sale)
    std_x_sale = np.reshape(std_x_sale, (1, len(std_x_sale)))
    std_y_sale = np.asarray(std_y_sale)
    std_y_sale = np.reshape(std_y_sale, (1, len(std_y_sale)))
    std_z_sale = np.asarray(std_z_sale)
    std_z_sale = np.reshape(std_z_sale, (1, len(std_z_sale)))

    len_entra = np.asarray(len_entra)
    len_entra = np.reshape(len_entra, (1, len(len_entra)))
    speed_entra = np.asarray(speed_entra)
    speed_entra = np.reshape(speed_entra, (1, len(speed_entra)))
    direction_entra = np.asarray(direction_entra)
    direction_entra = np.reshape(direction_entra, (1, len(direction_entra)))

    len_sale = np.asarray(len_sale)
    len_sale = np.reshape(len_sale, (1, len(len_sale)))
    speed_sale = np.asarray(speed_sale)
    speed_sale = np.reshape(speed_sale, (1, len(speed_sale)))
    direction_sale = np.asarray(direction_sale)
    direction_sale = np.reshape(direction_sale, (1, len(direction_sale)))

    #%% Correlacion con la media para los vehiculos que entran
    mean_data_entra = np.concatenate((mean_x_entra, mean_y_entra, mean_z_entra, len_entra, speed_entra), 0)
    mean_data_entra = np.transpose(mean_data_entra)

    mean_data_mod_entra = pd.DataFrame(mean_data_entra, columns=['mean_x_entra', 'mean_y_entra',
                                                                 'mean_z_entra', 'len_entra', 'speed_entra'])

    sns.pairplot(mean_data_mod_entra)
```

```
# Coeficientes de correlacion
mean_corr_matrix_entra = mean_data_mod_entra.corr()

#%% Correlacion con la desviacion estandar para los vehiculos que entran
std_data_entra = np.concatenate((std_x_entra, std_y_entra, std_z_entra, len_entra, speed_entra),
std_data_entra = np.transpose(std_data_entra)

std_data_mod_entra = pd.DataFrame(std_data_entra, columns=['std_x_entra', 'std_y_entra', 'std_z_entra',
                                                           'len_entra', 'speed_entra'])
sns.pairplot(std_data_mod_entra)

# Coeficientes de correlacion
std_corr_matrix_entra = std_data_mod_entra.corr()

#%% Correlacion con la media para los vehiculos que salen
mean_data_sale = np.concatenate((mean_x_sale, mean_y_sale, mean_z_sale, len_sale, speed_sale), 0
mean_data_sale = np.transpose(mean_data_sale)

mean_data_mod_sale = pd.DataFrame(mean_data_sale, columns=['mean_x_sale', 'mean_y_sale', 'mean_z_sale',
                                                           'len_sale', 'speed_sale'])
sns.pairplot(mean_data_mod_sale)

# Coeficientes de correlacion
mean_corr_matrix_sale = mean_data_mod_sale.corr()

#%% Correlacion con la desviacion para los vehiculos que salen
std_data_sale = np.concatenate((std_x_sale, std_y_sale, std_z_sale, len_sale, speed_sale), 0)
std_data_sale = np.transpose(std_data_sale)

std_data_mod_sale = pd.DataFrame(std_data_sale, columns=['std_x_sale', 'std_y_sale', 'std_z_sale',
                                                         'len_sale', 'speed_sale'])

sns.pairplot(std_data_mod_sale)

# Coeficientes de correlación
std_corr_matrix_sale = std_data_mod_sale.corr()


#%% PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(data_sensor)
principalDF = pd.DataFrame(data=principalComponents, columns = ['principal component 1',
                                                                'principal component 2'])
target = pd.DataFrame(np.round(np.reshape(y[:,2], (len(y[:,2]), 1)), decimals=3), columns=['target'])
finalDf = pd.concat([principalDF, target], axis = 1)

distintos = len(target)
for i in range(len(target)):
   for j in range(i+1, len(target)):
      if target['target'][i] == target['target'][j]:
         distintos -= 1
         break

# Para tamaño y velocidad
#fig = plt.figure()
#plt.xlabel('Principal Component 1')
#plt.ylabel('Principal Component 2')
#plt.title('Direccion')
#plt.scatter(finalDf['principal component 1'], finalDf['principal component 2'],
           # c=y[:,2], edgecolor='none', alpha=0.5,
           # cmap=plt.cm.get_cmap('nipy_spectral', distintos))
#plt.colorbar();

# Para direccion
targets = [0, 1]
colors = ['r', 'g']
fig = plt.figure()
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Direccion')
for targ, color in zip(targets,colors):
   indicesToKeep = finalDf['target'] == targ
   plt.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
      , finalDf.loc[indicesToKeep, 'principal component 2']
```

```
      , c = color
      , s = 50)
plt.legend(targets)
plt.grid()


#%% TSNE
X_embedded = TSNE(n_components=3).fit_transform(data_sensor)
principalDF = pd.DataFrame(data=X_embedded, columns = ['principal component 1',
                                                       'principal component 2'])
target = pd.DataFrame(np.round(np.reshape(y[:,2], (len(y[:,2]), 1)), decimals=3), columns=['target'])
finalDf = pd.concat([principalDF, target], axis = 1)

distintos = len(target)
for i in range(len(target)):
   for j in range(i+1, len(target)):
      if target['target'][i] == target['target'][j]:
         distintos -= 1
         break

# Para tamaño y velocidad
#fig = plt.figure()
#plt.xlabel('Principal Component 1')
#plt.ylabel('Principal Component 2')
#plt.title('Tamaño')
#plt.scatter(finalDf['principal component 1'], finalDf['principal component 2'],
           # c=y[:,1], edgecolor='none', alpha=0.5,
           # cmap=plt.cm.get_cmap('nipy_spectral', distintos))
#plt.colorbar();

# Para direccion
targets = [0, 1]
colors = ['r', 'g']
fig = plt.figure()
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Direccion')
for targ, color in zip(targets,colors):
   indicesToKeep = finalDf['target'] == targ
   plt.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
   , finalDf.loc[indicesToKeep, 'principal component 2']
   , c = color
   , s = 50)
plt.legend(targets)
plt.grid()
```

Anexo H. Código del clasificador

```
import numpy as np
import os
import random as rn
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import confusion_matrix as cm

# initialize variables
len_vehicles = []
speed_vehicles = []
direction_vehicles = []
data_sensor = []
num_samples = 20

# Carga el fichero con los datos del sensor con normalizacion gaussiana
# Cada patron: x,y,z,mean_x, std_x, mean_y, std_y, mean_z, std_z

local_path = 'G:/Mi unidad/TFM/aprendizaje/data_xyz_' + str(num_samples) + '_mean_std.csv'
data_aux = pd.read_csv(local_path,delimiter=',')
data_aux = data_aux.drop('Unnamed: 0', 1)
data_sensor = data_aux.get_values()

# Carga las etiquetas
# direccion : "0" sale, "1" entra
# tamaño: normalizacion
# velocidad: normalización

y = np.zeros((len(data_sensor),3))
local_path = 'G:/Mi unidad/TFM/aprendizaje/labels.csv'
labels_aux = pd.read_csv(local_path,delimiter=',')

y[:,0] = labels_aux['Tamagno'].get_values()
y[:,1] = labels_aux['Velocidad'].get_values()
y[:,2] = labels_aux['direccion'].get_values()

#%%
# Divide los datos en entrenamiento y test
x_train, x_test, y_train, y_test = train_test_split(data_sensor, y[:,2], test_size = 0.2)
x_train = np.reshape(x_train, (np.size(x_train,0), 1, np.size(x_train,1)))
x_test = np.reshape(x_test, (np.size(x_test,0), 1, np.size(x_test,1)))
y_train = np.reshape(y_train, (len(y_train),1))
y_test = np.reshape(y_test, (len(y_test),1))

#%%
learning_rate = 0.001 # Factor de aprendizaje
epochs = 500 # numero de iteraciones del algoritmo de entrenamiento
1
os.environ['PYTHONHASHSEED']=str(123456)
np.random.seed(123456)
tf.random.set_seed(123456)
rn.seed(123456)

def build_model():
    model = keras.Sequential([ layers.Dense(1, activation='sigmoid') ])
    optimizer = keras.optimizers.RMSprop(learning_rate)
    model.compile( loss = 'binary_crossentropy',
    optimizer = optimizer,
    metrics = ['binary_accuracy'])

    return model

model = build_model()

#%% Entrenamiento
history = model.fit(x_train, y_train, epochs = epochs)
hist = pd.DataFrame(history.history)
```

```
hist['epoch'] = history.epoch
hist.tail()

#%%
plt.figure()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Función de coste')
plt.plot(hist['epoch'], hist['loss'], label = 'Train Error')
plt.plot(hist['epoch'], hist['val_loss'], label = 'Val Error')

plt.figure()
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.plot(hist['epoch'], hist['binary_accuracy'], label = 'Train Error')
plt.plot(hist['epoch'], hist['val_binary_accuracy'], label = 'Val Error')

loss, acc = model.evaluate(x_test, y_test, verbose=2)
test_predictions = model.predict(x_test).flatten()

plt.figure()
plt.title('Valor predicho y valor real de los datos de test')
plt.xlabel('Datos')
plt.ylabel('Valores')
plt.plot(np.round(test_predictions), 'o',color='r')
plt.plot(y_test, 'x',color='b')
plt.legend(["Valor real", "Valor predicho"])
plt.show()

#%% Creacion de grupos equilibrados para la validacion cruzada

# Separar segun la direccion.
# Desordenarlos
# Coger 8 de uno y 8 de otro
# Obtener 10 grupos de 16 patrones

data_out = []
data_in = []
dir_out = []
dir_in = []

# Separa segun la direccion
for i in range(y.shape[0]):
    if y[i,2] == 0:
        data_out.append(data_sensor[i,:])
        dir_out.append(y[i,2])
    else:
        data_in.append(data_sensor[i,:])
        dir_in.append(y[i,2])

# Desordena los datos y etiquetas
data_dir_out = list(zip(data_out, dir_out))
data_dir_in = list(zip(data_in, dir_in))

rn.shuffle(data_dir_out)
rn.shuffle(data_dir_in)

data_out_shuffle, dir_out_shuffle = zip(*data_dir_out)
data_in_shuffle, dir_in_shuffle = zip(*data_dir_in)

# Crea 10 grupos equilibrados de 16 datos
groups = {'1': {'data_sensor': [], 'label': []},'2':{'data_sensor': [], 'label': []},
          '3':{'data_sensor': [], 'label': []},'4':{'data_sensor': [], 'label': []},
          '5':{'data_sensor': [], 'label': []},'6':{'data_sensor': [], 'label': []},
          '7':{'data_sensor': [], 'label': []},'8':{'data_sensor': [], 'label': []},
          '9':{'data_sensor': [], 'label': []},'10':{'data_sensor': [], 'label': []}}

for i in range(1, 11):
    if len(data_in_shuffle) >= 8*i:
        for j in range(8):
            groups[str(i)]['data_sensor'].append(data_out_shuffle[(i-1)*8 + j])
            groups[str(i)]['label'].append(dir_out_shuffle[(i-1)*8 + j])
        for j in range(8):
            groups[str(i)]['data_sensor'].append(data_in_shuffle[(i-1)*8 + j])
```

```
            groups[str(i)]['label'].append(dir_in_shuffle[(i-1)*8 + j])
        else:
            for j in range((i-1)*8,len(data_out_shuffle)):
                groups[str(i)]['data_sensor'].append(data_out_shuffle[j])
                groups[str(i)]['label'].append(dir_out_shuffle[j])
            for j in range((i-1)*8,len(data_in_shuffle)):
                groups[str(i)]['data_sensor'].append(data_in_shuffle[j])
                groups[str(i)]['label'].append(dir_in_shuffle[j])

        # Desordena los datos de los grupos
        aux = list(zip(groups[str(i)]['data_sensor'], groups[str(i)]['label']))
        rn.shuffle(aux)
        groups[str(i)]['data_sensor'], groups[str(i)]['label'] = zip(*aux)

    # Validacion cruzada

    current_fold = 1
    train_acc_hist = []
    test_acc_hist = []
    prediction = []
    targets = []

    for i in range(1, 11):
        KFold_X_test = np.asarray(groups[str(i)]['data_sensor'])
        KFold_X_test = np.reshape(KFold_X_test, (np.size(KFold_X_test,0), 1, np.size(KFold_X_test,1)
        KFold_Y_test = np.asarray(groups[str(i)]['label'])
        KFold_Y_test = np.reshape(KFold_Y_test, (np.size(KFold_Y_test,0), 1))

        aux_x = []
        aux_y = []

        for j in range(1, 11):
            if i == j:
                continue

            for k in range(len(groups[str(j)]['data_sensor'])):
                aux_x.append(groups[str(j)]['data_sensor'][k])
                aux_y.append(groups[str(j)]['label'][k])

        KFold_X_train = np.asarray(aux_x)
        KFold_X_train = np.reshape(KFold_X_train, (np.size(KFold_X_train,0), 1, np.size(KFold_X_train, 1)))
        KFold_Y_train = np.asarray(aux_y)
        KFold_Y_train = np.reshape(KFold_Y_train, (np.size(KFold_Y_train,0), 1))

        model = build_model()

        history = model.fit(KFold_X_train, KFold_Y_train, epochs=epochs, verbose=0)

        loss_train, acc_train = model.evaluate(KFold_X_train, KFold_Y_train, verbose=2)
        loss_test, acc_test = model.evaluate(KFold_X_test, KFold_Y_test, verbose=2)

        train_acc_hist.append(acc_train)
        test_acc_hist.append(acc_test)
        current_fold = current_fold + 1

        test_predictions = model.predict(KFold_X_test).flatten()
        prediction.append(test_predictions)
        targets.append(KFold_Y_test)

    mean_acc_train = np.mean(train_acc_hist)
    mean_acc_test = np.mean(test_acc_hist)
    std_acc_train = np.std(train_acc_hist)
    std_acc_test = np.std(test_acc_hist)

    print('Final train set k-fold cross validation accuracy: ', mean_acc_train , std_acc_train)
    print('Final test set k-fold cross validation accuracy: ', mean_acc_test , std_acc_test)

    targets = np.asarray(targets)
    targets = np.reshape(targets, (np.size(targets),1))
    prediction = np.asarray(np.round(prediction))
    prediction = np.reshape(prediction, (np.size(prediction),1))
    con_matrix = cm(np.asarray(targets), np.asarray(prediction))
```

Anexo I. Código del estimador de tamaño

```
import numpy as np
import os
import random as rn
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import confusion_matrix as cm

# initialize variables
len_vehicles = []
speed_vehicles = []
direction_vehicles = []
data_sensor = []
num_samples = 20

# Carga el fichero con los datos del sensor con normalizacion gaussiana
# Cada patron: x,y,z,mean_x, std_x, mean_y, std_y, mean_z, std_z

local_path = 'G:/Mi unidad/TFM/aprendizaje/data_xyz_' + str(num_samples) + '_mean_std.csv'
data_aux = pd.read_csv(local_path,delimiter=',')
data_aux = data_aux.drop('Unnamed: 0', 1)
data_sensor = data_aux.get_values()

# Carga las etiquetas
# direccion : "0" sale, "1" entra
# tamaño: normalizacion
# velocidad: normalización

y = np.zeros((len(data_sensor),3))
local_path = 'G:/Mi unidad/TFM/aprendizaje/labels.csv'
labels_aux = pd.read_csv(local_path,delimiter=',')

y[:,0] = labels_aux['Tamagno'].get_values()
y[:,1] = labels_aux['Velocidad'].get_values()
y[:,2] = labels_aux['direccion'].get_values()

#%%
# Divide los datos en entrenamiento y test
x_train, x_test, y_train, y_test = train_test_split(data_sensor, y[:,0], test_size = 0.2)
x_train = np.reshape(x_train, (np.size(x_train,0), 1, np.size(x_train,1)))
x_test = np.reshape(x_test, (np.size(x_test,0), 1, np.size(x_test,1)))
y_train = np.reshape(y_train, (len(y_train),1))
y_test = np.reshape(y_test, (len(y_test),1))

#%%
nh1 = 3 # numero de neuronas en la capa oculta 1
learning_rate = 0.001 # Factor de aprendizaje
epochs = 500 # numero de iteraciones del algoritmo de entrenamiento
os.environ['PYTHONHASHSEED']=str(123456)
np.random.seed(123456)
tf.random.set_seed(123456)
rn.seed(123456)

def build_model():
    model = keras.Sequential([
                              layers.Dense(nh1,activation='tanh'),
                              layers.Dense(1)
                             ])

    optimizer = keras.optimizers.RMSprop(learning_rate)
    model.compile( loss = 'mse',
                   optimizer = optimizer,
                   metrics = ['mape', 'mae'])
    return model

model = build_model()
```

```
#%% Entrenamiento
history = model.fit(x_train, y_train, epochs = epochs, validation_split=0.1)
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail()

#%%
plt.figure()
plt.xlabel('Epoch')
plt.ylabel('MAPE')
plt.plot(hist['epoch'], hist['mape'], label = 'Train Error')
plt.plot(hist['epoch'], hist['val_mape'], label = 'Val Error')

plt.figure()
plt.xlabel('Epoch')
plt.ylabel('MSE')
plt.plot(hist['epoch'], hist['loss'], label = 'Train Error')
plt.plot(hist['epoch'], hist['val_loss'], label = 'Val Error')

plt.figure()
plt.xlabel('Epoch')
plt.ylabel('MAE')
plt.plot(hist['epoch'], hist['mae'], label = 'Train Error')
plt.plot(hist['epoch'], hist['val_mae'], label = 'Val Error')

loss, mape, mae = model.evaluate(x_test, y_test, verbose=2)
test_predictions = model.predict(x_test).flatten()

plt.figure()
plt.scatter(y_test, test_predictions)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.axis('equal')
plt.axis('square')
plt.xlim([0,plt.xlim()[1]])
plt.ylim([0,plt.ylim()[1]])
_ = plt.plot([-100, 100], [-100, 100])

plt.figure()
plt.plot(y_test)
plt.plot(test_predictions)
plt.legend(["true", "pred"])
plt.show()

#%% Creacion de grupos equilibrados para la validacion cruzada

# Separar segun la direccion.
# Desordenarlos
# Coger 8 de uno y 8 de otro
# Obtener 10 grupos de 16 patrones

data_out = []
data_in = []
len_out = []
len_in = []

# Separa segun la direccion
for i in range(y.shape[0]):
    if y[i,2] == 0:
        data_out.append(data_sensor[i,:])
        len_out.append(y[i,0])
    else:
        data_in.append(data_sensor[i,:])
        len_in.append(y[i,0])

# Desordena los datos y etiquetas
data_len_out = list(zip(data_out, len_out))
data_len_in = list(zip(data_in, len_in))

rn.shuffle(data_len_out)
rn.shuffle(data_len_in)

data_out_shuffle, len_out_shuffle = zip(*data_len_out)
data_in_shuffle, len_in_shuffle = zip(*data_len_in)
```

```
# Crea 10 grupos equilibrados de 16 datos
groups = {'1': {'data_sensor': [], 'label': []},'2':{'data_sensor': [], 'label': []},
          '3':{'data_sensor': [], 'label': []},'4':{'data_sensor': [], 'label': []},
          '5':{'data_sensor': [], 'label': []},'6':{'data_sensor': [], 'label': []},
          '7':{'data_sensor': [], 'label': []},'8':{'data_sensor': [], 'label': []},
          '9':{'data_sensor': [], 'label': []},'10':{'data_sensor': [], 'label': []}}

for i in range(1, 11):
    if len(data_in_shuffle) >= 8*i:
        for j in range(8):
            groups[str(i)]['data_sensor'].append(data_out_shuffle[(i-1)*8 + j])
            groups[str(i)]['label'].append(len_out_shuffle[(i-1)*8 + j])
        for j in range(8):
            groups[str(i)]['data_sensor'].append(data_in_shuffle[(i-1)*8 + j])
            groups[str(i)]['label'].append(len_in_shuffle[(i-1)*8 + j])
    else:
        for j in range((i-1)*8,len(data_out_shuffle)):
            groups[str(i)]['data_sensor'].append(data_out_shuffle[j])
            groups[str(i)]['label'].append(len_out_shuffle[j])
        for j in range((i-1)*8,len(data_in_shuffle)):
            groups[str(i)]['data_sensor'].append(data_in_shuffle[j])
            groups[str(i)]['label'].append(len_in_shuffle[j])

    # Desordena los datos de los grupos
    aux = list(zip(groups[str(i)]['data_sensor'], groups[str(i)]['label']))
    rn.shuffle(aux)
    groups[str(i)]['data_sensor'], groups[str(i)]['label'] = zip(*aux)

#%% Validacion cruzada
current_fold = 1
train_mape_hist = []
test_mape_hist = []

for i in range(1, 11):
    KFold_X_test = np.asarray(groups[str(i)]['data_sensor'])
    KFold_X_test = np.reshape(KFold_X_test, (np.size(KFold_X_test,0), 1, np.size(KFold_X_test,1)
    KFold_Y_test = np.asarray(groups[str(i)]['label'])
    KFold_Y_test = np.reshape(KFold_Y_test, (np.size(KFold_Y_test,0), 1))

    aux_x = []
    aux_y = []

    for j in range(1, 11):
        if i == j:
            continue
        for k in range(len(groups[str(j)]['data_sensor'])):
            aux_x.append(groups[str(j)]['data_sensor'][k])
            aux_y.append(groups[str(j)]['label'][k])

    KFold_X_train = np.asarray(aux_x)
    KFold_X_train = np.reshape(KFold_X_train, (np.size(KFold_X_train,0), 1, np.size(KFold_X_train, 1)))
    KFold_Y_train = np.asarray(aux_y)
    KFold_Y_train = np.reshape(KFold_Y_train, (np.size(KFold_Y_train,0), 1))

    model = build_model()

    history = model.fit(KFold_X_train, KFold_Y_train, epochs=epochs, verbose=0)

    loss_train, mape_train, mae_train = model.evaluate(KFold_X_train, KFold_Y_train, verbose=0)
    loss_test, mape_test, mae_test = model.evaluate(KFold_X_test, KFold_Y_test, verbose=0)

    train_mape_hist.append(mape_train)
    test_mape_hist.append(mape_test)
    current_fold = current_fold + 1

    test_predictions = model.predict(KFold_X_test).flatten()

mean_mape_train = np.mean(train_mape_hist)
mean_mape_test = np.mean(test_mape_hist)
std_mape_train = np.std(train_mape_hist)
std_mape_test = np.std(test_mape_hist)
print('Final train set k-fold cross validation mape: ', mean_mape_train, std_mape_train)
print('Final test set k-fold cross validation mape: ', mean_mape_test, std_mape_test)
```

Anexo J. Código del estimador de velocidad

```
import numpy as np
import os
import random as rn
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import confusion_matrix as cm

# initialize variables
len_vehicles = []
speed_vehicles = []
direction_vehicles = []
data_sensor = []
num_samples = 20

# Carga el fichero con los datos del sensor con normalizacion gaussiana
# Cada patron: x,y,z,mean_x, std_x, mean_y, std_y, mean_z, std_z

local_path = 'G:/Mi unidad/TFM/aprendizaje/data_xyz_' + str(num_samples) + '_mean_std.csv'
data_aux = pd.read_csv(local_path,delimiter=',')
data_aux = data_aux.drop('Unnamed: 0', 1)
data_sensor = data_aux.get_values()

# Carga las etiquetas
# direccion : "0" sale, "1" entra
# tamaño: normalizacion
# velocidad: normalización

y = np.zeros((len(data_sensor),3))
local_path = 'G:/Mi unidad/TFM/aprendizaje/labels.csv'
labels_aux = pd.read_csv(local_path,delimiter=',')

y[:,0] = labels_aux['Tamagno'].get_values()
y[:,1] = labels_aux['Velocidad'].get_values()
y[:,2] = labels_aux['direccion'].get_values()

#%%
# Divide los datos en entrenamiento y test
x_train, x_test, y_train, y_test = train_test_split(data_sensor, y[:,1], test_size = 0.2)
x_train = np.reshape(x_train, (np.size(x_train,0), 1, np.size(x_train,1)))
x_test = np.reshape(x_test, (np.size(x_test,0), 1, np.size(x_test,1)))
y_train = np.reshape(y_train, (len(y_train),1))
y_test = np.reshape(y_test, (len(y_test),1))

#%%
nh1 = 3 # numero de neuronas en la capa oculta 1
learning_rate = 0.001 # Factor de aprendizaje
epochs = 500 # numero de iteraciones del algoritmo de entrenamiento
os.environ['PYTHONHASHSEED']=str(123456)
np.random.seed(123456)
tf.random.set_seed(123456)
rn.seed(123456)

def build_model():
    model = keras.Sequential([
                              layers.Dense(nh1,activation='tanh'),
                              layers.Dense(1)
                             ])

    optimizer = keras.optimizers.RMSprop(learning_rate)
    model.compile( loss = 'mse',
                   optimizer = optimizer,
                   metrics = ['mape', 'mae'])
    return model

model = build_model()
```

```
#%% Entrenamiento
history = model.fit(x_train, y_train, epochs = epochs, validation_split=0.1)
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail()

#%%
plt.figure()
plt.xlabel('Epoch')
plt.ylabel('MAPE')
plt.plot(hist['epoch'], hist['mape'], label = 'Train Error')
plt.plot(hist['epoch'], hist['val_mape'], label = 'Val Error')

plt.figure()
plt.xlabel('Epoch')
plt.ylabel('MSE')
plt.plot(hist['epoch'], hist['loss'], label = 'Train Error')
plt.plot(hist['epoch'], hist['val_loss'], label = 'Val Error')

plt.figure()
plt.xlabel('Epoch')
plt.ylabel('MAE')
plt.plot(hist['epoch'], hist['mae'], label = 'Train Error')
plt.plot(hist['epoch'], hist['val_mae'], label = 'Val Error')

loss, mape, mae = model.evaluate(x_test, y_test, verbose=2)
test_predictions = model.predict(x_test).flatten()

plt.figure()
plt.scatter(y_test, test_predictions)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.axis('equal')
plt.axis('square')
plt.xlim([0,plt.xlim()[1]])
plt.ylim([0,plt.ylim()[1]])
_ = plt.plot([-100, 100], [-100, 100])

plt.figure()
plt.plot(y_test)
plt.plot(test_predictions)
plt.legend(["true", "pred"])
plt.show()

#%% Creacion de grupos equilibrados para la validacion cruzada

# Separar segun la direccion.
# Desordenarlos
# Coger 8 de uno y 8 de otro
# Obtener 10 grupos de 16 patrones

data_out = []
data_in = []
len_out = []
len_in = []

# Separa segun la direccion
for i in range(y.shape[0]):
    if y[i,2] == 0:
        data_out.append(data_sensor[i,:])
        len_out.append(y[i,1])
    else:
        data_in.append(data_sensor[i,:])
        len_in.append(y[i,1])

# Desordena los datos y etiquetas
data_len_out = list(zip(data_out, len_out))
data_len_in = list(zip(data_in, len_in))

rn.shuffle(data_len_out)
rn.shuffle(data_len_in)

data_out_shuffle, len_out_shuffle = zip(*data_len_out)
data_in_shuffle, len_in_shuffle = zip(*data_len_in)
```

```
# Crea 10 grupos equilibrados de 16 datos
groups = {'1': {'data_sensor': [], 'label': []},'2':{'data_sensor': [], 'label': []},
          '3':{'data_sensor': [], 'label': []},'4':{'data_sensor': [], 'label': []},
          '5':{'data_sensor': [], 'label': []},'6':{'data_sensor': [], 'label': []},
          '7':{'data_sensor': [], 'label': []},'8':{'data_sensor': [], 'label': []},
          '9':{'data_sensor': [], 'label': []},'10':{'data_sensor': [], 'label': []}}

for i in range(1, 11):
    if len(data_in_shuffle) >= 8*i:
        for j in range(8):
            groups[str(i)]['data_sensor'].append(data_out_shuffle[(i-1)*8 + j])
            groups[str(i)]['label'].append(len_out_shuffle[(i-1)*8 + j])
        for j in range(8):
            groups[str(i)]['data_sensor'].append(data_in_shuffle[(i-1)*8 + j])
            groups[str(i)]['label'].append(len_in_shuffle[(i-1)*8 + j])
    else:
        for j in range((i-1)*8,len(data_out_shuffle)):
            groups[str(i)]['data_sensor'].append(data_out_shuffle[j])
            groups[str(i)]['label'].append(len_out_shuffle[j])
        for j in range((i-1)*8,len(data_in_shuffle)):
            groups[str(i)]['data_sensor'].append(data_in_shuffle[j])
            groups[str(i)]['label'].append(len_in_shuffle[j])

    # Desordena los datos de los grupos
    aux = list(zip(groups[str(i)]['data_sensor'], groups[str(i)]['label']))
    rn.shuffle(aux)
    groups[str(i)]['data_sensor'], groups[str(i)]['label'] = zip(*aux)

#%% Validacion cruzada
current_fold = 1
train_mape_hist = []
test_mape_hist = []

for i in range(1, 11):
    KFold_X_test = np.asarray(groups[str(i)]['data_sensor'])
    KFold_X_test = np.reshape(KFold_X_test, (np.size(KFold_X_test,0), 1, np.size(KFold_X_test,1)
    KFold_Y_test = np.asarray(groups[str(i)]['label'])
    KFold_Y_test = np.reshape(KFold_Y_test, (np.size(KFold_Y_test,0), 1))

    aux_x = []
    aux_y = []

    for j in range(1, 11):
        if i == j:
            continue
        for k in range(len(groups[str(j)]['data_sensor'])):
            aux_x.append(groups[str(j)]['data_sensor'][k])
            aux_y.append(groups[str(j)]['label'][k])

    KFold_X_train = np.asarray(aux_x)
    KFold_X_train = np.reshape(KFold_X_train, (np.size(KFold_X_train,0), 1, np.size(KFold_X_train, 1)))
    KFold_Y_train = np.asarray(aux_y)
    KFold_Y_train = np.reshape(KFold_Y_train, (np.size(KFold_Y_train,0), 1))

    model = build_model()

    history = model.fit(KFold_X_train, KFold_Y_train, epochs=epochs, verbose=0)

    loss_train, mape_train, mae_train = model.evaluate(KFold_X_train, KFold_Y_train, verbose=0)
    loss_test, mape_test, mae_test = model.evaluate(KFold_X_test, KFold_Y_test, verbose=0)

    train_mape_hist.append(mape_train)
    test_mape_hist.append(mape_test)
    current_fold = current_fold + 1

    test_predictions = model.predict(KFold_X_test).flatten()

mean_mape_train = np.mean(train_mape_hist)
mean_mape_test = np.mean(test_mape_hist)
std_mape_train = np.std(train_mape_hist)
std_mape_test = np.std(test_mape_hist)
print('Final train set k-fold cross validation mape: ', mean_mape_train, std_mape_train)
print('Final test set k-fold cross validation mape: ', mean_mape_test, std_mape_test)
```